Theory and Experiments in Autonomous Sensor-Based Motion Planning with Applications for Flight Planetary Microrovers

Thesis by
Sharon Lynn Laubach

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1999

(Defended May 24, 1999)

Acknowledgements

A long time ago in a hometown far, far away...a youngster was entranced by the 1977 movie *Star Wars*, in which two of the most beloved of the major characters were "droids": fully autonomous robots which interacted with their world independently of human (or other species') control. As futuristic as these droids still seem today, R2D2 and C-3PO shaped the public's conception of robotics—and mine.

A few years later, as a sixth-grader, I wrote to the various NASA centers asking for information about Jupiter and the other outer planets. In return, NASA/JPL sent me a glorious full-colour pamphlet filled with images of the king of planets, returned by Voyager in the midst of its continuing mission. I was enthralled by the prospects of robotic space exploration.

Thus I am indebted in no small part to George Lucas and to NASA/JPL for inspiring a young explorer determined to see exotic worlds through the eyes of autonomous robots she helped fashion.

Since those halcyon days, the pathway to my goal has been long and tortuous. During my tenure at Caltech, I have been helped by many people along the way, and I am indebted to each of them. Although I am unable to thank them all here, I reassure those not mentioned that I have not forgotten.

First and foremost, I thank my advisor, Dr. Joel W. Burdick, without whom the redemption of my Caltech experience would not have been possible. He has been a tremendous supporter, motivator, source of inspiration, and sounding board, and I deeply appreciate his guidance and his willingness to give so much of his time and energy to his students. In addition, I appreciate his support for my decision to accept the invitation to join the Mars Pathfinder Rover flight team as a "rover driver" (officially, Sequence Planner). Despite taking time away from my studies, the Pathfinder experience proved invaluable on many levels, including giving me firsthand experience with semi-autonomous navigation (and all of its relevant issues) on the surface of another planet. Finally, I would like to thank Joel for his concern and support during my particularly difficult final month of graduate school.

I would also like to thank Dr. Samad Hayati and the Long Range Science Rover

(Rocky7) Team at the Jet Propulsion Laboratory (JPL) for their support during this endeavour, including sponsoring my research. The opportunities to work close-at-hand with an actual prototype rover, to implement my algorithms on that rover, to wrestle with system issues, and finally to see my algorithms succeed in real Mars-simulating terrains, contributed immensely to my success. I particularly appreciate the help of Dr. J. "Bob" Balaram, Dr. Clark Olson, Steve Peters, Richard Petras, and Dr. Richard Volpe. I also thank Todd Litwin at JPL for his always patient help with understanding complex vision processing code, and Bill Adler and Mike Allen for their quick response to computer needs (such as recovering mistakenly deleted files).

I especially treasure the opportunity to work with the Mars Pathfinder Rover Flight Team. They gave me a once-in-a-lifetime experience I will never forget, and introduced me to a band of rogues I look forward to working with as colleagues and friends. I particularly thank Dr. Bob Anderson, Brian Cooper, Dr. Justin Maki, Dr. Jake Matijevic, Jack Morrison, Tam Nguyen, Dr. Tim Parker, Al Sirota, Dr. Henry Stone, Jan Tarsala, Art Thompson, Dr. Rick Welch, and Brian Wilcox, for making my time in Mission Control especially memorable, and for their help long after the mission was over. In addition, I thank Mars Pathfinder Project Scientist Dr. Matt Golombek for his explanation of Mars rock densities.

Throughout all of my time at JPL, Dr. Larry Matthies has been an inspiration, a ready sourcebook for information, and a friend who has helped me through some of my most difficult trials. I cannot thank him enough for his willingness to set aside time in his ever-overloaded schedule for me whenever I have needed his help, even serving as a member of my defense committee and giving his time and energy in a careful reading of a draft of this dissertation.

At Caltech, many people have influenced my work and me. I thank Dr. John Hauser for discussing esoteric mathematics with me. Dr. Mark Long and Dr. Bill Goodwine were terrific officemates (when I was actually around campus). I am indebted to Drs. Rod Goodman, Pietro Perona, and Demetri Psaltis for serving on my defense committee. I am grateful for those who helped me through my first

years at Caltech: Dr. John Doyle gave me a start, and Drs. Richard Murray, Jorge Tierno, and Robert Bodenheimer, Jr., helped me find my way. I would like to thank Maria Koeper for all of her cheerful help and for those afternoon chats while waiting for the Big Guy. I am also grateful to Wendy McKay for introducing me to Textures (near-WYSIWYG TEX on the Mac!) and to Marlene Hagen for helping to usher me through the last harried weeks of graduate school.

Along the way, I have had many, many friends who have buoyed my spirits and have helped keep me going along my path. I am most grateful for the constant support of Marc Pack, who has seen me through everything grad school has thrown at me, and who is always ready with his quick wit to deter the blues. A big thank you also goes to Monica Hubbard; I was a member of her Caltech Women's Glee Club for almost my entire stay at Caltech, and appreciated the opportunity for a few hours each week to concentrate on making music and to forge lasting friendships with my fellow singers. I would also like to thank Dr. Eugenia Kuo and Lara Hughes; they both began as my Catalina roommates and ended up among my most valued friends.

I am thankful for my family and their unwavering support throughout my extended scholastic career. My mother, Bobbie, and my brother, Jonathan, were always proud of me, and were a source of strength during my tribulations. I also deeply appreciate the support of my grandparents, Mary and Bronson, and of my Uncle Don. Granddaddy in particular has always made sure I was down to earth, and I have enjoyed showing him just how useful NASA is for everyday people.

Finally, I would like to thank my fiancé, Andrew H. Mishkin. Serving as my coach, he helped spur my research on, and since then he has aided me both technically—as the Mars Pathfinder Rover lead uplink engineer and system engineer on the 2001 and 2003/2005 Mars rover missions—and personally. I am grateful for all of the time and effort he put into helping me with my work, and for his understanding—and ready bouquets of snapdragons—when grad school weighed heavily upon me. I am looking forward to returning the favour with help on his own manuscript.

Theory and Experiments in Autonomous Sensor-Based Motion Planning with Applications for Flight Planetary Microrovers

by

Sharon Lynn Laubach

In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Abstract

With the success of Mars Pathfinder's Sojourner rover, a new era of planetary exploration has opened, with demand for highly capable mobile robots. These robots must be able to traverse long distances over rough, unknown terrain autonomously, under severe resource constraints. Much prior work in mobile robot path planning has been based on assumptions that are not truly applicable to navigation through planetary terrains. Based on the author's firsthand experience with the Mars Pathfinder mission, this work reviews issues which are critical for successful autonomous navigation of planetary rovers. No current methodology addresses all of these constraints. We next develop the sensor-based "Wedgebug" motion-planning algorithm. This algorithm is complete, correct, requires minimal memory for storage of its world model, and uses only on-board sensors, which are guided by the algorithm to efficiently sense only the data needed for motion planning, while avoiding unnecessary robot motion. The planner has the additional advantage of producing locally-optimal paths, and is suitable for robots with a field-of-view limited in both downrange and angular scope, for a variety of applications including planetary navigation. This work includes the proof of completeness and correctness of the Wedgebug algorithm, and in particular provides a corrected, detailed proof of a key result required for the proof of completeness of the Wedgebug algorithm (and for the TangentBug algorithm which inspired this approach). In addition, we

extend this result to a broader class of environments. The implementation of a version of Wedgebug, called "RoverBug," on the Rocky7 Mars Rover prototype at the Jet Propulsion Laboratory (JPL) is described, and experimental results from operation in simulated martian terrain are presented.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

APXS	Alpha Proton X-ray Spectrometer
AU	Astronomical Unit (1 AU = distance from the Earth to the Sun)
BF	boundary following (mode in Wedgebug algorithm)
bps	bits per second
C-space	configuration space
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
DOE	Department of Energy
DOF	Degree of Freedom
DSN	Deep Space Network
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIDO	Field Integrated Design & Operations (2003 rover prototype)
FOV	Field of View
g.c.p.	generalised critical point
HMMWV	High Mobility Multi-Wheeled Vehicle
IMP	Imager for Mars Pathfinder
JPL	Jet Propulsion Laboratory
LBAL	Lander-Based Autonomous Localisation
LCD	Liquid Crystal Display
LRSR	Long Range Science Rover
LTG	Local Tangent Graph
MAE	Material Adhesion Experiment
MAV	Mars Ascent Vehicle

MIPS	Million Instructions Per Second
MLST	Mars Local Solar Time
MPF	Mars Pathfinder
MtG	motion-to-goal (mode in Wedgebug algorithm)
NASA	National Aeronautics and Space Agency
NavCam	Navigation Camera (instrument on the 2003/05 rovers)
OPP	Opportunistic Path Planner
PanCam	Panoramic Camera (instrument on the 2001/03/05 missions)
PROM	Programmable Read-Only Memory
RF	Radio Frequency
RHU	Radioisotope Heating Unit
RTG	Radioisotope Thermoelectric Generator
SCE	Sequencing and Command Executive
SGI	Silicon Graphics, Inc.
sol	martian day
UHF	Ultra High Frequency
VL1	Viking Lander 1
VL2	Viking Lander 2
WEB	Warm Electronics Box
WITS	Web Interface for Telescience

Chapter 1

Laying the Course

You have ability to over-
come obstacles on
the way to success.
—from a fortune cookie

It is rather hard work: there is now no smooth road into
the future: but we go round, or scramble over the obstacles.
—D. H. Lawrence, *Lady Chatterley's Lover*

1.1 Introduction

In science fiction, robots are tools, characters, objects of fear or of tremendous aid—and nearly always autonomous. Although today's researchers are still far from achieving writers' dreams of fully autonomous machines, significant advances have been made in recent years, particularly in the area of robotic motion planning. Improved, cost-effective sensors have increased the information that robots can learn about their environment, spurring interest in sending robots to places humans cannot, will not, or should not go: hazardous waste sites, damaged nuclear reactors, deep ocean trenches, and deep space.

Such missions—into areas which are unexplored, and about whose layouts only sketchy information may exist—are greatly facilitated by robots which are able to not only sense their environs, but to react accordingly as they navigate toward their goal. *Sensor-based motion planning* [9], as a field, arose to address this issue. The basic sensor-based "find-goal" problem is formulated as follows: the robot is situated

in unknown terrain, usually with an associated coordinate frame. The robot is given a goal (a coordinate point), generally outside of its current sensor range. Using only its own on-board sensors for gathering information, the robot must detect and avoid obstacles as it moves incrementally toward the goal.

The urgency of adding autonomous motion-planning capabilities to robots rises dramatically in the case of planetary exporation. (See Chapter 2 for an extensive discussion of this subject.) The requirements of newer missions are increasingly unattainable without the aid of mobile robots utilising advanced autonomous techniques. However, these missions are still subject to the severe constraints of flight hardware, such as power, mass, and cost, which in turn affect the available sensors, memory, and computational power. These topics are treated in detail in Chapter 2, and constitute the primary motivation for the work presented in the subsequent chapters. We note that the basic sensor-based problem applies here, with the caveat that the algorithm must be tailored to fit within the available computational framework and the given sensor suite. In addition, the robot is expected to provably achieve the designated goal, so resources (including time) are not wasted correcting errors in final robot position before the mission can continue.

Although this work is strongly motivated by space applications, the resulting algorithm can be used to advantage in a variety of scenarios, such as hazardous or radioactive waste cleanup, ocean floor exploration, etc., where sensors and environmentally shielded computational prowess are at a premium. Given a sensor array with very limited coverage, both angular and downrange, the robot must conserve both sensor queries (expensive in memory and processing of sensor data) and robot motion (expensive in positioning error and possibly energy). The robot must represent detected obstacles simply, to conserve both memory and computational effort. And finally, the robot must successfully avoid obstacles while maintaining provable incremental progess toward the goal, through unbounded, rough terrain, until the goal is achieved. The Wedgebug algorithm, developed in this thesis, fulfills all of these supplemental requirements for the sensor-based *rover "find-goal" problem*, with the additional property that the resulting paths are *locally optimal*. Locally

optimal paths consist of segments which are optimal as measured by path length, when only obstacles able to be sensed by the robot along that segment are considered. As the range and angular coverage of the robot sensors approach infinity and 360 degrees, respectively, the locally optimal path approaches the globally optimal solution in many cases, particularly if the robot sensors are able to "see over" the surrounding obstacles to rover mobility. The name of the algorithm, Wedgebug, derives from its relation to the Tangent Bug algorithm developed by Kamon, Rimon, and Rivlin [23] in 1995, whose name in turn acknowledges the globally-convergent, reactive nature of the "Bug" algorithms first developed by Lumelsky and Stepanov in 1987 [40]. These algorithms, and their exact relation to Wedgebug, are discussed in more detail in Section 1.2.

The related RoverBug algorithm, also developed in this thesis, is the implementation of the Wedgebug concept on an actual vehicle, the Mars Rover prototype, Rocky7. RoverBug extends Wedgebug to the case when the robot sensor array is capable of imaging regions beyond the immediate obstacles, and assimilates various idiosyncracies of the Rocky7 system. The RoverBug program has been tested extensively in rough, naturalistic terrain, and has demonstrated outstanding performance.

1.2 Historical Overview

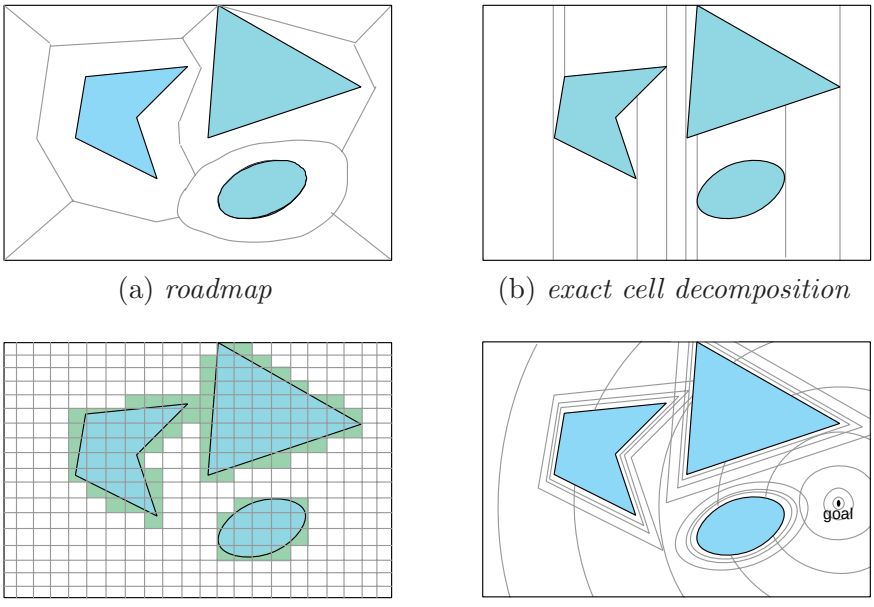
We first present a brief overview of the major concepts in, and historical categories of, robot motion planning. Later, we review related work in the specific area of *rover* motion planning.

1.2.1 Robot Motion Planning

Motion planning as a field can be generally split by three guiding philosophies: *classical path planning*, *heuristic planning*, and *"complete and correct" sensor-based path planning*. In order to show where Wedgebug fits in this hierarchy, we discuss each philosophy in turn, along with the associated advantages and disadvantages of each

approach. In the discourse which follows, the robot is assumed to be modelled as a point located at x in a two-dimensional environment*, and is capable of holonomic motion. Obstacles block the robot's sensors as well as its motion. The *free space*, $\mathfrak{F}$, comprises the allowable regions for robot traversal: the 2D environment, minus the (interiors of) the obstacles. Finally, unless otherwise specified, the robot's sensors have 360° angular coverage and limited range, R.

Classical motion planners assume that full knowledge of the geometry of the robot's environs is known *a priori*. However, the classical planners have the useful properties of *correctness* and *completeness*. A path is *correct* if it lies wholly within $\mathfrak{F}$, and, if the goal is reachable, connects the robot's initial position with the goal. Whereas any useful path planner produces correct paths, completeness is a highly desirable virtue: the planner will generate a path if one is possible, and will halt otherwise in finite time. Latombe describes classical planners in some detail in his book, *Robot Motion Planning* [36], in which he splits the classical planners



(a) *roadmap* (b) *exact cell decomposition*

(c) *approx. cell decomposition* (d) *potential field*

Figure 1.1: Examples of types of "classical" planners

*The 2D workspace is chosen for simplicity. Most of the techniques described here extend to higher dimensional spaces.

into three major categories: roadmap algorithms, cell decomposition methods, and potential field approaches. The first two categories seek to create maps or channels for robot navigation (see Fig. 1.1(a)). Examples of roadmap algorithms include the visibility and reduced visibility (also known as tangent) graphs [57],[36], Voronoi graphs [58], and Canny's silhouette method [7]. Each of these schemes constructs a set of one-dimensional curves which encapsulate the topology of the free space, and serve as a system of "freeways," complete with methods for entry and exit, to enable the robot to traverse from start to goal. Cell decomposition algorithms, on the other hand, come in two varieties: those which break $\mathfrak{F}$ into an exact polygonal decomposition (Fig. 1.1(b)), and approximate techniques which overlay a regular grid (with possible local adaptations in resolution) on the entire world model (Fig. 1.1(c)). Examples of the exact variety include trapezoidal and algebraic decompositions [36]; grid methods include applications of A*, and quadtree decompositions [36]. Whereas the potential field methods (Fig. 1.1(d)), unlike the other types of planners just described, do not explicitly map preferred routes—they act instead as heuristics to guide the search of a grid laid over the configuration space of the robot—the original philosophy still held that the entire world model was known. In addition, though Khatib's original formulation of the potential field method did not possess the property of completeness [27], further development of the theory has produced a classical potential field algorithm which is complete [66].

As previously stated, the classical planners as a group possess the key advantages of provable completeness and correctness. Furthermore, since the world model is known *a priori*, allowing these algorithms to be computed "off-line" (that is, the entire path can be computed before the robot ever moves, and there is no dependence upon the robot's sensors), in general the computational complexity of the classical planners can be analysed. The cell decomposition methods have the additional advantage that they produce safe corridors between obstacles, rather than hard-to-follow one-dimensional curves. However, classical planners are often impractical to implement, relying for example on geometric properties not able to be sensed easily or at all by the robot as it moves, or demanding excessive computational effort.

Even more damning (considering the "rover problem") is the fact that classical planners require complete knowledge of the environment beforehand, and that often the environment must be bounded to guarantee completeness.

The class of heuristic planners, such as Brooks' subsumption architecture [5] or the track arbitration schemes developed at CMU [26],[78], as well as the "Go To Waypoint" algorithm employed by the Sojourner rover [54] and the Rocky7 rover [82] (see Table 1.1), share the useful property of being able to be made sensor-based much more easily than the classical planners, and can be applied to unknown terrains. These planners dispense with the idea of creating global models of the environment in favor of "using the world as its own model" and using only local knowledge of the environs to inform the robot's reactions, usually chosen from a set of "behaviours." However, although heuristic planners are designed to work well in "most" environment configurations, they lack completeness—there is no guarantee that the algorithm will halt, or that the robot will be able to find the goal even if a path exists. In addition, the heuristic planners tend to result in lengthy paths in natural terrain (relative to the optimal path), as has been seen in field trials with the Rocky7 rover using the "Go To Waypoint" algorithm [37]. (Most traverses of the Sojourner rover on Mars have not been long enough to demonstrate this effect.)

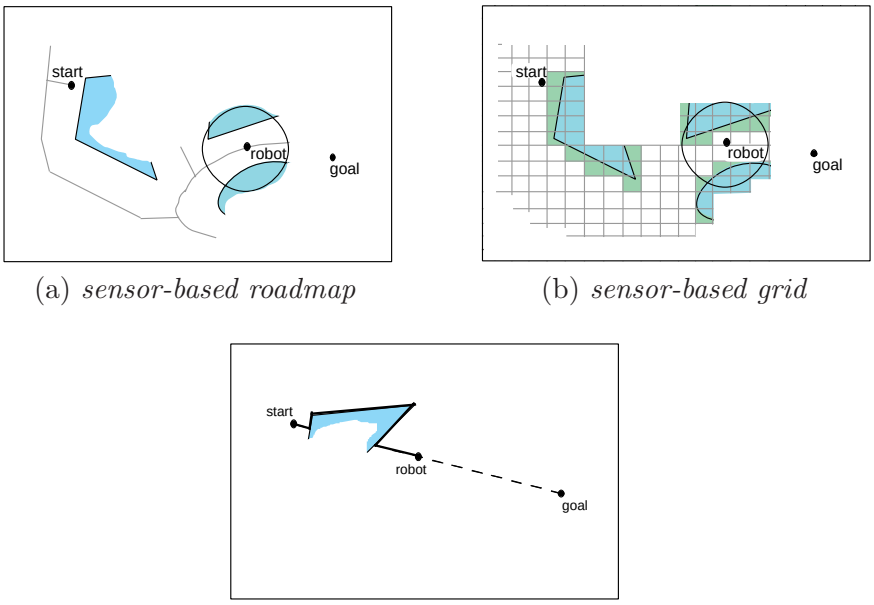
It is desirable, then, to develop a path planner which combines the best of both worlds: good local properties with a global convergence criterion. Such planners fall under the aegis of what can be called "complete and correct" sensor-based motion

"Go To Waypoint" Behaviours:

Turn-to-Goal	turn in place towards goal (unless at goal)
Obstacle Detect	measure terrain in front of rover
Turn-in-Place	turns away (using nominal rotation angle) from detected obstacles
Thread-the-Needle	attempts to move between obstacles to left & right; backs up if alley is later blocked
Loop-toward-Goal	based upon difference between orientation & goal direction, arcs toward goal (choosing from three arc radii) for nominal distance

Table 1.1: Example of a heuristic planner

planning, the most relevant of the three categories to the problem of autonomous planetary motion planning. These algorithms are incremental in nature: the robot senses its environment, then determines a local path segment based upon the resultant world model. After moving along the local path, the robot begins the cycle again with its sensors. Using this model, three distinct approaches have been explored, two of which adapt classical methods to a local sensed region. One set of methods incrementally builds "roadmaps" within the free space in the visible area, such as Choset's HGVG [9] and Rimon's adaptation of Canny's OPP [65]. Of note is the "Tangent Bug" algorithm, developed by Kamon, Rivlin, and Rimon [23]. The second approach springs from approximate cell decomposition, filling in a grid-based world model incrementally, such as Stentz' D* algorithm [78], [77]. The third approach harks more closely to the heuristic planners, and includes the "Bug" algorithms of Lumelsky and Stepanov [40], [64], which combine reactive behaviours with global parameters to reach the goal. All of these methods maintain provable properties of correctness and completeness, yet are fully applicable to unknown terrains. Both heuristic planners and these sensor-based planners share the



(a) *sensor-based roadmap* (b) *sensor-based grid*

(c) *Bug algorithm*

Figure 1.2: Examples of types of "sensor-based complete & correct" planners

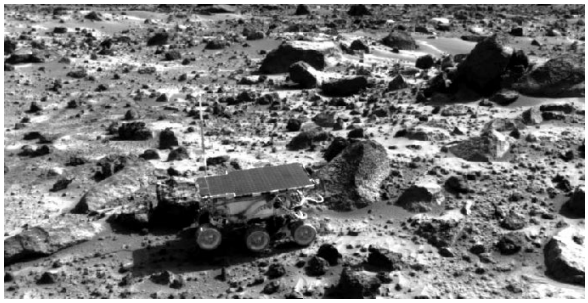


Figure 1.3: Typical terrain encountered on Mars by the Sojourner rover. The intrepid mobile explorer is 68cm long by 48cm wide, and stands 28cm tall.

disadvantage that their computational complexity is difficult to analyse, primarily due to the algorithms' reliance upon sensor input for decision-making. For this same reason, both types of planners are subject to sensor error, and it is uncertain how such errors affect the performance of many of the methods. In particular, several schemes rely upon "good" (or "perfect") dead reckoning ability.

The above sensor-based methods have each been developed to differing degrees in their application to real systems. For example, the sensor-based version of OPP is currently strictly theoretical, owing to the difficult-to-implement nature of the sensors required. The HGVG, on the other hand, has been implemented on a mobile robot using range sensors. Choset's planner produces paths which are maximally distant from obstacles, a plus for rover safety. However, it works best in contained and cluttered environments, especially those with well-defined corridors; a description not applicable to the typical martian environment (see Fig. 1.3), and lacks gaze control or other means to reduce sensing effort.

Both D* and TangentBug are useful in unbounded environments, and produce "locally optimal" solutions, that is, the paths are the shortest length possible using solely local information). D* in particular has been implemented on a real world system (an autonomous HMMWV driven in a slag heap near Pittsburgh). However, the grid-based world model requires a significant amount of memory for storage, and the algorithm's completeness depends entirely upon the precision of its world model, which is determined by cell granularity. D* also contains no provision for

minimising the amount of sensing needed. The Wedgebug algorithm, an extension of the Tangent Bug concept, is a "complete and correct" sensor-based planner which shares the advantages of Tangent Bug while being applicable to the harsh constraints of the rover "find-goal" problem.

We now turn our attention to prior work which has been done specifically in the area of rover motion planning.

1.2.2 *Rover* Motion Planning

The literature on motion planning for rovers, or more generally for outdoor, rough terrain (or "off-road") mobile robots, includes work designed for and/or implemented on a wide variety of vehicles, ranging from large HMMWV's fitted with sensors, computers, and actuators, to large prototype planetary rovers (e.g., Robby at JPL and the Ames/CMU rover Nomad, both comparable in size to a small car), to the small-size microrovers which have flown to Mars and are being developed for future missions. We summarise below the properties of the Wedgebug algorithm, the focus of this thesis, and next describe several highlights from the recent literature, and discuss aspects of the selected algorithms relevant to motion planning for flightlike microrovers.

The Wedgebug algorithm is complete and correct, and produces locally-optimal (shortest distance) paths through unbounded, unknown terrain. This technique also utilises gaze control to minimise the amount of sensing, while avoiding unnecessary robot motion. Its world model is local and simple, yielding the dual benefits of not bookkeeping a global model (with all of its attendant registration issues), and minimising memory requirements. A version of the algorithm has been implemented on the Rocky7 prototype microrover.

Much of the prior work in the area of rover navigation has concentrated on heuristic local trajectory generation, and particularly obstacle avoidance. This style of navigation has also been referred to as "local navigation" [79] and "piloting" [43], especially when paired with a goal-seeking behaviour. Examples include:

- Gat's implementation of his ATLANTIS architecture (1991) [12], utilises a

heuristic planner based on Slack's "navigation templates" [75] for obstacle avoidance and goal seeking, as well as a version of Barraquand and Latombe's non-holonomic planner [4] to recover from cul-de-sacs. The planner was implemented on JPL's Robby, a large prototype rover. This planner is claimed to be complete, though to the author's knowledge, no proof has been presented.

- the Autonomous Cross-Country Navigation (ACCN) algorithm of Brumitt, Coulter, and Stentz (1992) [6], implemented on the Navlab II (a HMMWV), which is used to follow a given global path by selecting control points within the visible region along the path, then generating a *control path* and simulating a traverse along that path. If a potential collision is detected, up to two new control paths are generated by adding control points on either side of the encountered obstacle. In the case that a dead-end is encountered, the vehicle stops and the run is terminated. This algorithm was designed for fast, continuous driving along a predetermined path in fair terrain; a situation not applicable to the upcoming Mars microrover missions.

- Langer, Rosenblatt, and Hebert's behaviour-based system (1994) [35], using Ganesha to manage the local map grid and to detect untraversable cells, and DAMN to arbitrate between votes on steering directions contributed by a set of behaviours. For off-road navigation, these behaviours include obstacle avoidance, which votes against steering arcs which pass through untraversable cells or which constitute a "near miss" of an obstacle; and goal seeking, which uses pure pursuit to generate a preferred turn radius. This system has also been implemented on a HMMWV, and shares the ACCN system's susceptibility to dead-ends (i.e., the algorithm is not complete).

- RANGER, developed in Singh and Kelly (1995) [74] and Kelly and Stentz (1997,8) [24], [25]. This system, implemented on both a HMMWV and CMU's prototype lunar rover RATLER, also arbitrates between inputs from hazard avoidance and goal-seeking behaviours. It optimises the behaviour of following a given goal trajectory (or pure pursuit of a goal position), while using hazard

avoidance as a constraint. The hazard avoidance behaviour simulates traverses along a collection of candidate local trajectories (corresponding to steering angles) over an elevation grid, taking into account the vehicle dynamics on the sensed terrain. Although this system is able to handle traversability issues, clearly simulation based upon models of both the local terrain and of the vehicle require significant computational resources, not available on the current batch of flightlike Mars microrovers.

- Pagnot and Grandjean's fast cross-country navigation approach (1995) [60], which computes a (typically trapezoidal) "trace" of possible rover trajectories (including uncertainty in sensing, motion control, and localisation) in each of 20 directions, incremented by 1° steps centered about the direction toward the goal. Each trace ends when it encounters an obstacle. The planner then selects the trace which brings the rover closest to the goal, favouring a choice which places obstacles to the sides of the path rather than in front. This algorithm incorporates a primitive form of gaze control: if the resulting path is too short, the rover is directed to aim its sensors 20° from the current heading, to the left or right depending on the direction of the initial (too short) path. After three such cycles, the algorithm halts. This system has been implemented on the IARES prototype planetary rover and tested in the GEROMS Mars/Lunar analogue site at CNES. The algorithm is designed for mostly flat, uncluttered terrains.

- the two versions of the Mid-Course Navigation technique developed by Nakatani and Kubota et al. (1995-98) [70], [56], [32]. This approach, implemented on a testbed rover using a laser range finder (which cannot see beyond impassable obstacles), calculates a potential field on a local elevation grid, then uses a "tracking seeds" search method (simulating seeds, sown along the farthest (obstacle free) grid cells from the current position, floating along the potential flow) to generate local candidate subpaths, from which the optimal path (minimising the potential along the path) is chosen. These methods also include

a form of automatic gaze control: if no forward path can be found, the rover will sense a region to the left and right of the current visible patch.

- the Go-to-Waypoint algorithm developed at JPL (1995) [47], [82], [54] and implemented on the Rocky3.2 and Rocky7 prototype microrovers, as well as on the Sojourner rover now on Mars. This algorithm cycles through a list of behaviours (see Table 1.1) in order to avoid obstacles and move toward the goal. A particular advantage of this system is that it is tuned to the severe computational and sensing contraints of the JPL class of microrovers.

Significantly, each of the heuristic approaches mentioned above lack the property of completeness. That is, although these algorithms are designed to work "well enough" within their respective domains, they are not guaranteed to reach the goal, nor to halt in finite time if the goal is unattainable.

Besides the heuristic techniques, several graph-search methods have been explored. For example:

- the D* method developed by Stentz (1993) [77], [78]. This method (described in Section 1.2.1) has been implemented on a HMMWV and on a prototype rover. It has recently been extended by Yahja, et al. (1998) [89] to a framed-quadtree version, in order to reduce the memory requirements of the world model and to solve the inability of D* to plan some straight diagonal paths. In addition, the D* planner has been implemented as part of a comprehensive system on a HMMWV, combined with a local navigator, SMARTY (similar to the Langer, Rosenblatt, and Hebert navigator described above), and the DAMN voting arbitrator [79]. D* has been proven to be resolution-complete[†] in a bounded environment [76].

- Chatila and Lacroix' 2D navigation algorithm (1995) [8], which has been implemented on the Adam prototype rover. From the sensed information, the

[†]That is, D* is complete, assuming the resolution of its world model grid is fine enough to avoid masking passages between obstacles. Note that the memory requirements of a grid increase as its resolution increases.

planner classifies the traversability characteristics of each sensed grid cell, which is then fused into a global bitmap model. Using this model, obstacles are first grown, and then regions with similar traversability characteristics are extracted. Next, nodes are defined along the regions' borders, and a graph search method (based upon energy, time, terrain class, and terrain labelling confidence) selects an optimal path. As a final step, perception tasks (for localisation and further terrain classification) are planned along this path.

- the breadth-first, parallel grid search method developed by Gennery (1998) [13]. This algorithm uses a parallel search method derived from Witkowski [88] on an extended elevation grid which includes information on slope and terrain roughness at each cell. At the end of the search, the total cost of the optimum path through each node in the local grid is known. As a result, it is easy to adjust the resultant path in order to produce a smoother path, for example. The algorithm also includes a notion of the probability that a given cell is untraversable, which is integrated over the candidate path; if the resultant probability exceeds a threshold, the path is terminated. The portion of the path before termination is executed, and a new sensing cycle begins. (In order to accomodate goals outside of the immediate area, the backward cost array for the edge(s) of the grid in the direction of the goal are appropriately initialised.)

Again, with the exception of Stentz' D* planner, none of these approaches have been proven complete.

A few approaches in the rover motion planning literature have not been implemented to the author's knowledge, but have been demonstrated in simulation:

- the similar extended elevation grid methods developed by Simeon and Dacre Wright (1992) [73] and by Kubota et al. (1995,7) [33], [34]. Both of these methods use a grid search method to select a path, based upon simulated static placements of a given rover model on the terrain at each node. These methods assume full prior knowledge of the terrain.

- Shiller, Gwo, and Chen's B-spline optimal planner (1990) [72], which models the terrain as a B-patch, generates the best K initial paths by graph search using kinematic constraints, then filters these paths using dynamic constraints and proximity to another path. Finally, the remaining paths are optimised according to time or distance. This method, as with the extended elevation grid methods, assumes full prior knowledge of the terrain.

- A heuristic algorithm which significantly differs from those described above is NAFBA, developed by Martin-Alvarez (1994) [43]. This approach uses a fuzzy grid representation of the terrain, including height, roughness, and soil physics parameters. The map also contains representations of obstacles and of landmarks, used for localisation. A fuzzy grid search method, which incorporates notions of traversability cost as well as of the set of allowable "piloting behaviours" (e.g., "climb_L," "follow_contour_L," "move_to_P"), returns a program consisting of a series of desired piloting behaviours in addition to checkpoints for localisation. This method "never generates optimal paths" [43], but is able to handle some traversability issues, and generates linguistic descriptions of paths. Sensing issues—e.g., how to generate the fuzzy grid, and how to handle local *vs* global information—are not addressed.

In addition to not addressing sensor issues, each of these approaches has clear drawbacks in terms of computational resources required.

Of all of these approaches, only one is tuned to the severe computational, memory, and sensing constraints of flightlike planetary microrovers: the JPL "Go-to-Waypoint" algorithm. As discussed previously, this planner has the significant disadvantage that it is not complete—it is not guaranteed to reach the goal nor to halt in finite time. Although it showed good performance for short (up to 10m) traverses in somewhat cluttered terrain (0.57 obstacles per m^2), it is not necessarily suitable for much longer traverses or for more crowded environments [47]. Thus, it would be of great advantage to have a complete, correct motion planner, appropriate for a flightlike microrover, for future missions which place greater demands on rover

autonomy.

1.3 Contributions of this Work

A key contribution of this work is the development of an autonomous sensor-based motion planner for robots with limited field-of-view, named Wedgebug. The Wedgebug algorithm is complete, correct, produces locally optimal (shortest-length) paths (considering only the visible obstacles), and utilises automatic gaze control to both minimise the amount of sensing required and to avoid unnecessary robot motion. This planner comprises two major modes, "motion-to-goal" and "boundary following," which interact to ensure that the algorithm will cause the robot to converge to the goal (or to detect that the goal is unreachable, and thus halt) in finite time. In addition, these major modes are split into several submodes, including "virtual" submodes which invoke gaze control processes, in order to improve efficiency. This thesis also contains the proof of completeness and correctness for this novel algorithm. The algorithm is tuned to the severe constraints of flightlike microrovers: the world model is streamlined, consisting of obstacle vertices, and is not retained between subpaths, providing the benefits both of practical memory requirements and of not requiring the planner to register local maps (thus avoiding the issue of small uncertainties in sensing and localisation). Also, the algorithm is amenable to varying levels of autonomy, from single-subpath execution under tight operator guidance to complete autonomous traverse to distant goals. Besides being useful for microrovers, the algorithm could be applied to such scenarios as environmental hazard cleanup or military surveillance.

Another contribution of this work is the implementation of a version of Wedgebug, called RoverBug, on an actual prototype planetary microrover. The vehicle in question, the Rocky7 microrover, was designed and built at the Jet Propulsion Laboratory in order to aid in the development and testing of technologies for future planetary rover missions. As such, the Rocky7 rover is an ideal testbed for practical motion planners for flightlike planetary robots, as it clearly demonstrates many of

the severe constraints faced by such planners (detailed in Chapter 2).

A third contribution is the delineation of the issues and constraints faced by practical motion planners for flightlike planetary microrovers. Although these issues are currently being factored into the development of flight systems and software for future planetary missions, to the author's knowledge they have not been described in detail in any publication. Furthermore, the newness of the application (planetary flight microrovers), coupled with its timeliness and relevancy (Mars microrover missions are planned for every launch opportunity between 2001 and 2005, and potentially continuing into the future), renders such an exposition particularly valuable for future research.

Finally, this work contributes a proof of a key result required for the verification of completeness for the Wedgebug algorithm (and for the TangentBug algorithm which inspired this approach). The original proof, provided in [21] by Kamon, Rimon, and Rivlin, was slightly flawed, and further, was restricted to bounded workspaces populated by a finite number of obstacles, each with a smooth boundary. In this thesis, we not only correct the proof and remove the restriction to bounded environments, but also extend the result to obstacles with finitely piecewise smooth boundaries. Next, we introduce the concept of a "generalised critical point," and extend the original result into the domain of obstacles with finitely piecewise C^1, rectifiable boundaries.

1.4 Outline/ Conceptual Roadmap

The rest of this dissertation proceeds as follows: Chapter 2 describes in some detail the practical application, planetary exploration, which primarily inspired this work, and will give motivation for choices made in developing an appropriate path planner. Chapter 3 reviews several basic concepts in motion planning, and expounds the original TangentBug algorithm as developed by Kamon, Rivlin, and Rimon. The chapter continues with a discussion of the advantages of this particular approach in light of the requirements of a planetary mission, and points out where further

development is needed. A discussion of a key result and its proof, required for the proof of completeness for both TangentBug and Wedgebug, concludes the chapter. Chapter 4 develops the Wedgebug algorithm and introduces the concept of using gaze control to conserve robot motion. The chapter includes a proof of the method's completeness. Chapter 5 then shifts to implementation issues. The chapter begins with a description of the Jet Propulsion Laboratory's Rocky7 prototype Mars rover, the platform on which an extended version of the Wedgebug algorithm described above, called "RoverBug," was coded and tested. Next, we discuss several issues which arose during implementation, and discuss the changes incorporated into the RoverBug algorithm. Finally, we include the results of tests run with Rocky7 in the MarsYard, JPL's outdoor rover testing facility. The thesis' final chapter contains concluding remarks.

Chapter 2

Exploration and Mars Rovers

2.1 Exploring Beyond Earth

Since even before the time of the ancient Greeks, mankind has been fascinated by the stars, and by the velvety blackness between. As astronomy progressed, we learned that the stars were spread over vast reaches of space, and that some of those "stars" were actually distinct worlds, orbiting our same Sun. Scientists and writers alike dreamed about what those worlds were like, what lessons they might hold for those of us on Earth. In the latter part of this past century, we have gone even further: to devise methods to actively explore those worlds.

In our quest to probe the frontiers of the space around us, we have always been limited by the range of our senses. However, we have combated this limitation with ingenuity, by devising means to artificially extend our sensory reach. In particular, we have augmented the capabilities of Earth-based instruments to gather information about the planets, with spacefaring exploratory craft equipped with remote sensing technologies.

These planetary explorers have evolved through four basic phases of development.* In each phase, the types of knowledge which could be gained from planetary

*In the descant which follows, we will refer specifically only to the progression of the American unmanned space program, and that only as it applies to planetary exploration. In particular, we disregard the timeline and development of lunar exploration and of the Soviet unmanned space program. Both of these efforts followed a slightly different path, seeking first to impact the target body before developing (soft) landers and finally orbiters. It should be noted that in general, the Soviets were the first to impact/land on/orbit the

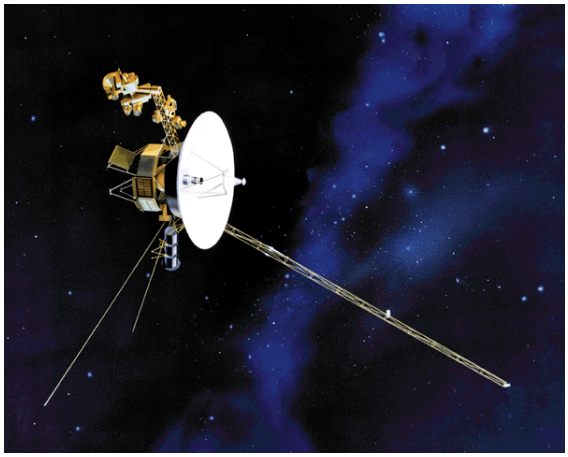


Figure 2.1: Artist's conception of the Voyager flyby spacecraft in flight. (courtesy JPL)

encounters have been dictated by the technologies available, which in turn drove both spacecraft design and the instruments carried by the craft. It should be noted that there is actually no "cut-off" time denoting the end of a particular phase: indeed, several types of spacecraft can be used simultaneously for a single mission. However, new phases of development bring with them new approaches to planetary exploration, enabling scientists to bring home new types of information about distant worlds.

In the 1960's through the mid 70's, these exploratory spacecraft were designed to fly by their target planets, gathering as much information as possible before the spacecraft whizzed past, perhaps angling toward another target. This class of spacecraft includes such examples as the Mariner missions to Mars, and the wildly successful Voyager "Grand Tours" to the outer planets (Fig. 2.1). Fly-by missions strive to gather "global" information: images of the planet as a whole, images of large-scale features, traces of the magnetosphere, and atmospheric or general

moon and the nearby planets, if sometimes only by a few months. For example, the Soviet mission Luna 9 landed on the Moon in 1966 (anticipating Surveyor 1's success by four months), followed by the world's first unmanned sample return mission, by Luna 16 in 1970. 1970 also featured the first unmanned rover, carried to the Moon by Luna 17. In contrast, although the USSR was the first to land on and orbit Mars (in 1971) and Venus (in 1970 (surface), 1975 (orbit)), samples have not yet been returned from any planet, and the first non-lunar unmanned rover was on the US Mars Pathfinder mission in 1997. [81]

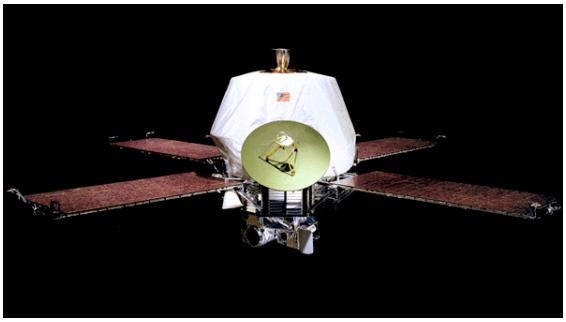


Figure 2.2: Photo of the Mariner 9 orbiting spacecraft. (courtesy JPL)

planetary chemical composition, for example.

The second phase began in the early 1970's with the development of spacecraft designed to orbit their target planets, which "proved to be a sensible stepping stone from flybys to a lander."[30] The first of these was Mariner 9 (Fig. 2.2), which entered orbit around Mars on November 14, 1971. These craft, such as the Viking orbiters around Mars, Magellan about Venus (Fig. 2.3), Galileo around Jupiter and today's Mars Global Surveyor, are meant to stay in operation for extended periods around a single planet. While in orbit, these spacecraft refine information gathered by flybys: while the earlier spacecraft were capable of sensing only spot patches on the target planet as they flew past, the orbiters allow scientists to integrate data from all over the planet, pulling together diverse regions' characteristics to formulate a coherent picture. In addition, since the orbiters fly over a given area repeatedly, they are able to not only provide greater detail for the area than is possible with a single pass, but can observe detailed changes over time, a true advance over flybys. For example, the data returned from Mariner 9 revolutionised scientists' image of Mars, originally formed by data from Mariners' 5 and 6 flyby missions [30].

Figure 2.3: Artist's conception of the Magellan orbiter at Venus. (courtesy JPL)

Neither type of spacecraft, however, is able to return very detailed information about the planet surface. Even today, the images received from Mars orbiters is rarely better than 300 meters/pixel, with only isolated "postage stamp" regions achieving the highest resolution of 1.4 m/pixel [50]. Thus, it is necessary to send a spacecraft actually to the surface to gather detailed information about a given site. This new era of exploration was ushered in by the Viking martian landers, in 1975 (Fig. 2.4). The Viking landers—besides imaging their environs in great detail—could physically manipulate the neighboring rocks and soil, helping to determine the substrate's mechanical properties; and could gather local samples for *in situ* chemical and biological analysis. Further, since the landers operated for years, they could monitor local seasonal changes in temperature, atmospheric composition, etc. The new information which can be provided only by surface instruments can again challenge hypotheses formulated from flyby and orbiter data: information gleaned from the Viking landers, for example, altered the thrust of the American martian program from biology to geology.

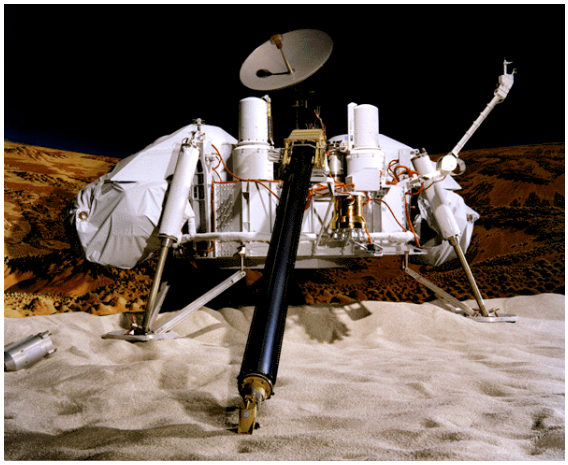


Figure 2.4: Photo of the Viking lander in a Mars diorama.
(courtesy JPL)

The advent of landers allowed scientists to do "close-up" science, as they are accustomed to doing on Earth. However, landers have a distinct shortcoming in that they are limited to a single site for study: to sample the elemental composition of an interesting pebble just beyond arm's reach, for example, is frustratingly impossible

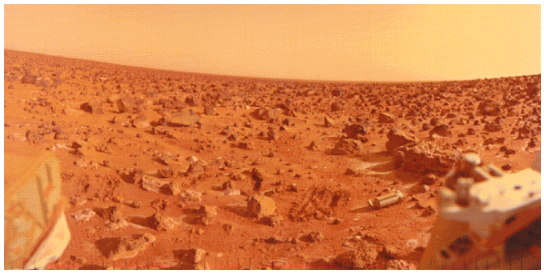


Figure 2.5: The trenches dug in the center of this image show the extent of the Viking lander's arm's reach. The lander is unable to explore the rest of the vast area it can see! (courtesy JPL)

without sending a second entire spacecraft (see Fig. 2.5). Nor is a lander capable of answering the question of just what lies beyond that tantalising ridge. The Mars Pathfinder mission in July, 1997, addressed these issues by carrying—along with a lander fitted with an imager and atmospheric instrument suite (Fig 2.6)—the first mobile robot to roam over the surface of another planet. The Sojourner rover (Fig. 2.7) marked the first success of the fourth phase of planetary exploration: a separate, mobile spacecraft which was capable of traversing the planetary surface; imaging features far from the lander—or even completely hidden from the lander's view—in great detail; and placing its instruments and conducting *in situ* experiments on samples in a variety of terrains well outside the lander's reach. The immediate promise of this new advance in planetary exploration—borne out by the plans for the next missions to Mars during the upcoming decade, as well as the 2002 nanorover mission to an asteroid, and proposed "rover" missions to Europa and to Titan[†]—

Figure 2.6: The Pathfinder lander on Mars, imaged by the Sojourner rover.

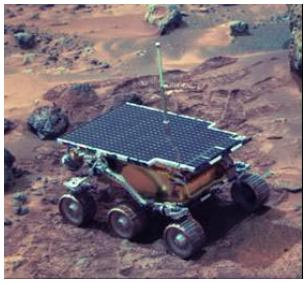


Figure 2.7: The Sojourner rover on Mars, imaged by the Pathfinder lander.

(both images courtesy JPL)

[†]The proposed missions in both of these cases may include variations on the "rover" concept: an independently mobile submarine; or a transformable "aerover," part rover, part balloon.

is that the capabilities of well-instrumented landers will become mobile, allowing scientists to learn about a much larger area of the planetary surface in detail, and enabling scientists to choose from a much larger collection of samples to return to Earth than would be practical with a network of stationary landers. It is likely, based upon experience with past phases of planetary exploration, that the results returned from these upcoming rover missions will again bring a new understanding of the alien worlds around us.

2.2 Spacecraft Operations: Challenges of a Planet

Beyond the problems solved in designing and building these diverse spacecraft is the question of how to operate them once they are in space. Consider the famous Voyager missions. As reported in the missions' status report on November 17, 1998 (paraphrased here):

> Voyager 2 is currently departing the solar system at a speed of 15.9 kilometers per second (35,000 miles per hour), and is now 8.4 billion kilometers (5.2 billion miles) from Earth, or more than 56 times farther from the Sun than Earth is. Round-trip light time from Earth to Voyager 2 is currently about 16 hours. Its twin, Voyager 1, is the most distant human-made object in space, at 10.8 billion kilometers (6.7 billion miles) from Earth, travelling at a speed of about 17.3 kilometers per second (38,752 miles per hour).[83]

We marvel at how these—and other—spacecraft can be accurately commanded over millions or billions of miles. However, the key for spaceborne craft navigation is precisely that their milieu is empty, predictable, deep space.

Spacecraft meant to remain in space, such as flyby and orbiter missions, can be commanded very precisely from millions of miles away because their deep space environment can be accurately modelled. Most hazards are known, with known trajectories and properties (such as gravitation) which might affect the spacecraft. Other potentially hazardous properties of the environment, such as temperature and radiation, are known and generally stable. Thus, not only can the spacecraft's trajectory and environment be plotted far enough in advance to ameliorate the communications delay caused by the finite speed of light, but in fact the mission

requirements can be plotted months in advance. Thus, command sequences for these spacecraft can be generated well ahead of when they apply, and only rarely do special circumstances arise which demand immediate attention from operators. Similarly, the trajectories of landers can be modelled quite accurately, from their cruise stage through space through entry, descent, and landing; the command sequences governing these stages are also created months in advance and tested thoroughly in simulation before use by the spacecraft.

Once on the surface, this operational model begins to break down. Landers, confined to a single site, still enjoy some measure of predictability in their environment, particularly after any peculiar properties of the landing site are learned. The environment is more hostile, however, than the deep space environment in many ways. For example, on the surface of a planet, the spacecraft no longer has continual solar power.[‡] A related problem—both are due to the planet's rotation—is thermal cycling. Components intended for surface operation must endure daily drastic temperature swings. For example, the Pathfinder rover measured extreme external temperatures from $-85.9°$C to $19.2°$C; and extreme internal temperatures from $-36.4°$C to $48.3°$C during its 83 sols of operation in the martian summer[§] [41]. In addition, surface-based spacecraft are subject to the weather patterns of their host planet; in the case of Mars, the most severe weather hazard is dust storms, which could potentially slash available solar power and abrade delicate instruments. The harsh surface environment limits the expected life of a landed spacecraft. As a result, reactive planning—rather than the deliberative planning characteristic of spaceborne craft—becomes more important, in order to squeeze more science out of the available time.

[‡]Access to solar power is important in the current political climate, which frowns upon the use of RTGs (Radioisotope Thermoelectric Generators) in spacecraft. It is as yet unclear how this resistance to the use of RTGs will affect future missions to the outer planets, where solar energy will not support spacecraft operation.

[§]The Pathfinder mission site is in the mid-latitudes on Mars. Internal temperature was recorded inside the WEB (Warm Electronics Box); external temperatures were measured at the wheels and in the MAE (Material Adhesion Experiment) on the solar panel. The temperature swing implied between day and night is real, and harsh.

The ability to react to the environment becomes crucial when mobile rovers are added. Besides dealing with the same hostile environment as landers, rovers encounter their own unique hazards. Unlike the case of deep space craft, the hazards associated with rover navigation are on a scale approaching the size of the vehicle itself (rover hazards are measured in centimeters, rather than kilometers). The substrate itself is variable, with properties which are unknown beforehand, which affects (among other aspects) the rover's ability to track its own position. Rocks may shift under a wheel, a cleat may catch on a buried cinder and slip: all with the result that the ultimate outcome of rover motion is, to some extent, unpredictable. Thus, these mobile surface spacecraft require the highest level of reactiveness to their environment in order to operate safely and to achieve their scientific goals.

2.3 Mission Constraints

There are three major constraints driving the design of surface spacecraft, and especially rovers: communications, power, and mass. Communications is clearly more difficult for surface craft than for deep space craft, due to the hostile environment encountered on the surface (temperature swings in particular adversely affect the operation of frequency-controlling crystals) and since the Earth is not always within line of sight of the spacecraft. The power available to the spacecraft is also a critical element, since it drives everything from which motors, instruments, radio elements, down to which CPU can be used. The rover must be able to carry its own power sources—a combination of solar arrays and batteries. Finally, every aspect of the spacecraft design must conform to a strict mass budget. Due to the current political climate, the "flagship missions" of the 1970's through early 1990's (e.g., Voyager, Viking, Mars Observer, Galileo, and concluding (forever?) with Cassini) are out of favour, replaced by the "faster, better, cheaper" credo. As a result, mass is even more stringently constrained, since mass translates directly into cost: the cost of the launch vehicle required to send a spacecraft into space.[¶] The 2003 sample return

[¶]The current rough cost of launching a pound into space is $>$ \$10,000 for low Earth orbit.

rover, for example, has a mass budget of roughly 70 kg (154 lbs). A fourth constraint, though perhaps not as obvious, is volume, since the rover must fit inside the surface delivery system, usually a lander with instruments in its own right, inside the launch vehicle fairing. One consequence of these constraints is that the rover cannot be heavily instrumented for navigation, since a higher priority is given to preserving mass, power, and volume for the science payload. Similarly, navigation concerns cannot choke the communication channel: the rover cannot be teleoperated by any means, due both to the communications delay from sheer distance and to communications issues including the lack of continual line of sight to Earth and the limited availability of the Deep Space Network, used to communicate with all planetary/deep space spacecraft. In addition, precious communication time spent transmitting navigation data reduces the amount of science data which can be sent.

Another, overarching constraint is the project budget. Spacecraft design, including software; fabrication, procurement (including the extremely high cost of flight-qualified electronics)[‖], and assembly; the launch vehicle; and operations must all fit within the cost cap. Thus, although rovers require intensive supervision, the budget precludes the proverbial "standing army" of operators, particularly for extended missions. This constraint, coupled with the issues detailed above, indicate that a method must be found to ensure that the rover is sufficiently responsive to its environment without requiring that the vehicle be explicitly commanded by Earth-based operators. The goal, then, is on-board rover autonomy. Ideally, in the long run, we will be able to send fully autonomous mobile spacecraft out across a planetary surface; they would report back occasionally with interesting new discoveries and experimental results.

[‖] The cost of environmentally-shielded electronics is staggering: "The price of a satellite could potentially double [for using radiation-hardened Pentium chips]."[61] Even so, rad-hard Pentium chips (the subject of a new DOE initiative) will not be available for four years, too late for incorporation into the next decade of Mars rovers. Current rad-hard CPUs are "generations behind the Pentium"[61], drilling home the need for appropriately tailored algorithms.

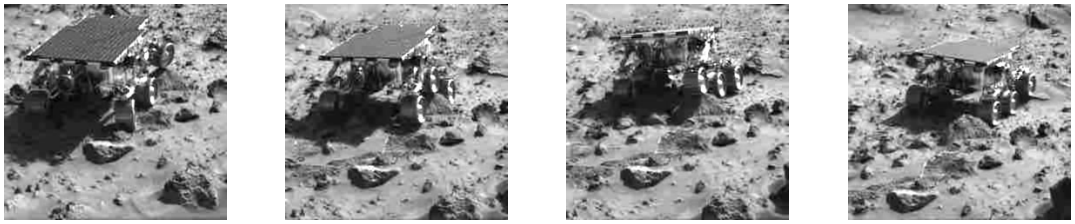
Figure 2.8: Sojourner's farthest traverse from the Sagan Memorial Station, accomplished on Sol 74. The rover's radial distance from the lander is 12.336 m. Sojourner is behind the region known as the Rock Garden; Chimp and Snoopy are the large rocks in the foreground. (courtesy JPL)

2.4 The State of the Art

The state of the art for flight autonomy for planetary rovers is the Sojourner rover, carried to the surface of Mars by the Mars Pathfinder (MPF) spacecraft on July 4, 1997 [42], [71]. This rover is known to have operated for a total of 83 sols (martian days), and was healthy when the lander fell silent on September 27, 1997**, thus surviving almost three times her designed "extended mission" lifetime. Sojourner traversed a total of 101.756 m, reaching a maximal distance of 12.336 m from the lander (distances are measured from rover odometry). (See Figs. 2.8, 2.12.) She returned more than 245 Mbits of data, including 193 images (57 in colour) and APXS (Alpha Proton X-ray Spectrometer) spectra from 16 distinct targets (9 rocks and 7 soil sites) [41]. Her mission is a resounding success, but the Sojourner rover still represents only the first step towards the goal of autonomy. We present her description to illustrate the design tradeoffs which will impact future flight rovers' autonomous capabilities.

Standing 68 cm x 48 cm x 28 cm, and weighing a mere 10.5 kg, the Sojourner rover is roughly the same size as a microwave oven. She features a 6 wheel drive, rocker-bogie suspension, which enables the rover to surmount obstacles 1.5 wheel

**September 27 marks the last day that data was received. The last communication of any kind with the lander was on October 7, 1997. Since Sojourner must communicate with Earth through the lander, the loss of signal ended the rover mission as well.



(a) (b) (c) (d)

Figure 2.9: Traverse over obstacle, Sol 24 (courtesy JPL)

diameters, or nearly 20 cm, tall[††] (see Fig. 2.9). (To put these numbers in perspective, this level of mobility is analogous to driving a car over a dining room table.) Her top speed is roughly 0.7 cm/sec. Besides carrying several experiments, including the aforementioned APXS, a material adhesion experiment on the solar panel, and an abrasion experiment on one wheel, the rover sports three cameras: two front-mounted black and white for navigation, and a rear-mounted colour science imager. (Of course, images returned from the front cameras proved important for science, as well, since Sojourner is able to return close-up images of features far from the lander. (see Fig. 2.10) The wheels also proved useful for soil mechanics experiments. (see Fig. 2.11))

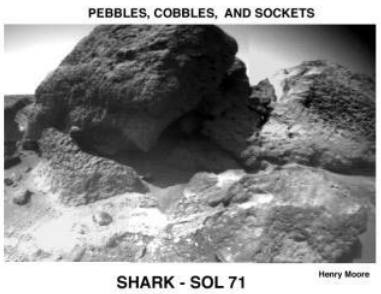


Figure 2.10: Close-up image returned by Sojourner, showing details of rock morphology. The unexpected pebbles and sockets suggest that liquid water was present to form these rocks, originally thought to be purely volcanic [62]. (courtesy JPL)

Sojourner's primary source of power is her 0.22 m^2 solar array, consisting of 13 strings of 18 gallium arsenide cells, with a peak output of roughly 16 W (at 17 V). Backup power, also used for nighttime operations, is provided by 3 strings of D-cell size lithium thionyl chloride nonrechargeable batteries. These batteries could supply up to 150 W-hours of energy (at 8-11 V)[‡‡] [41]. Sojourner's entire power budget

[††]3% of the ground in the MPF site was covered by rocks >20 cm tall, on average. The maximum concentration was 20%, in the "Rock Garden" [14],[16]. The predicted coverage was roughly 8% [15]; for comparison, VL1 had 4% coverage, and VL2 11% [85], [86].

[‡‡]The batteries, used during the mission primarily for nighttime operation and for on-demand power boosts during traverses, lasted 56 sols (thanks to careful conservation by

Figure 2.11: The rover digging, to characterise soil mechanics.
(courtesy JPL)

for computing is 3.5 W. Due both to these severe power restrictions and to the need for radiation-hardened, flight-proven circuitry, Sojourner is equipped with an Intel 80C85 processor operating at 2 MHz.*

The batteries, microprocessor, and other temperature-sensitive components are shielded from the harshest temperature swings within the Warm Electronics Box (WEB). The WEB is approximately 33 cm x 26.5 cm x 14.2 cm† (i.e., slightly larger than a shoebox), and features SiO_2 aerogel insulation and RHUs (Radioisotope Heating Units) as well as electrical heaters to keep the electronics within survivable temperatures ($\pm 40^\circ$ C). (Heating is also provided by the powered circuit boards themselves.) Also contained within the WEB is Sojourner's UHF radio modem, hardwired for 9600 bps‡ asynchronous communication with the lander (and through the lander, with Earth). As a result of the tight quarters, the rover's circuit boards are very densely populated and must be hand-designed to fit in their allotted space. Besides power, these volume constraints also limit the amount of on-board memory§, due to the large packages required for environmental shielding. Sojourner features 576 KB total volatile memory: 512 KB bulk RAM and 64 KB rad-hard storage; and 176 KB total non-volatile: 160 KB EEPROM, 16 KB shielded PROM. Clearly, these types of severe power and volume restrictions, as well as the high cost of

operators) before the rover switched to solar-only mode.

*The primary drivers for choice of CPU are environmental survivability, and a tradeoff between power draw and functionality [84].

†external dimensions

‡In comparison, current home computer modems are 56000 bps.

§The primary drivers for memory are the same as for CPU choice, plus cost (e.g., a single shielded PROM costs $30K) and volume [84].

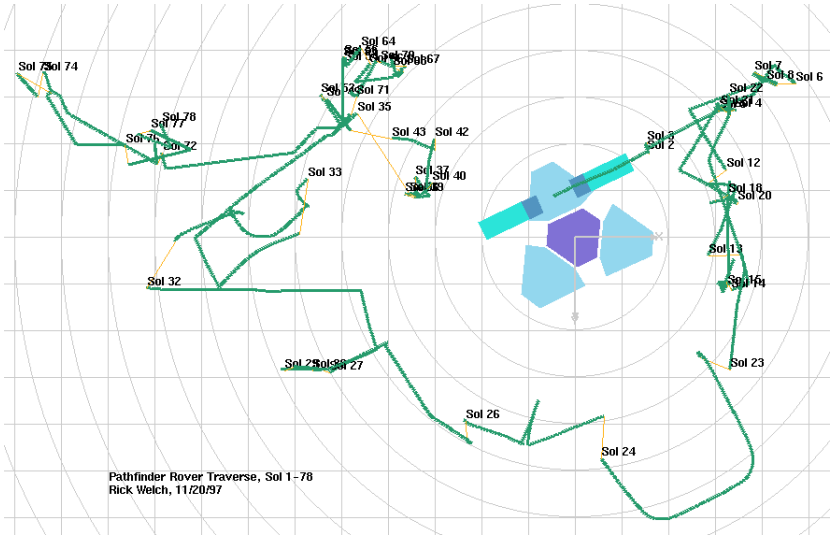


Figure 2.12: Map of Sojourner's traverse, from rover odometry, Sols 1-78 [87]. (courtesy Rick Welch)

flight-qualified electronics, will continue to constrain the computing capabilities—and therefore the complexity of on-board autonomy—of future rovers. Of note, in order to maximise use of her limited computational facilities, the Sojourner rover has no on-board operating system, and her software is written in C and assembly. It is expected that future rovers will have an on-board operating system, further restricting available memory.

As noted earlier, communications constraints preclude teleoperation of this "mobile geologist." Instead, ground-based rover operators (on Earth) enjoyed usually one or two communication opportunities per sol, during which the Deep Space Network (DSN) transmitted command sequences for the rover (and lander) and received buffered telemetry data. A typical rover sequence contains 200-300 commands, detailing everything from thermal control parameters, to health status check rates, to actual instrument operation and traverse instructions. These sequences are generated laboriously by the rover operators, based upon received telemetry.[¶] The

[¶]The author had the opportunity to witness the performance of the Sojourner rover firsthand during the first few weeks after Pathfinder landed on Mars, in her capacity as an uplink engineer on the Rover team. (Officially, her position was Mars Pathfinder Rover Sequence Reviewer, later Sequence Planner.) The Rover Flight Team comprised two distinct roles: downlink and uplink. The downlink engineers' task was to interpret data returned

traverse commands, in particular, necessitate an intensive building process: the designated "rover driver" dons LCD shuttered goggles in order to scrutinise stereo imagery returned from the lander's IMP (Imager for Mars Pathfinder) cameras. The driver utilises a 3D graphic icon modelled on the rover to both visually localise the rover within the lander coordinate frame, and to plan Sojourner's movements. In times when caution is necessary, the plan includes such detailed instructions as "turn clockwise 17.2 degrees, then move forward 10.8 cm" to manoeuver the robot through tight spaces. It should be noted that errors in rover sequences are dangerous, since—more so than with other spacecraft, whose environments are generally more predictable—a wrong command could potentially damage the rover.

In order to ease the burden on the ground-based operators—and to test the feasibility of the approach—the Sojourner rover features limited autonomous navigation ability, encapsulated primarily in the "Go To Waypoint" command [54]. Ground operators specify a goal location, and the rover moves toward the goal without further instruction, avoiding obstacles and other hazards on its own. Although the two navigation cameras have been partially calibrated for stereo, the rover relies upon a laser-striping system to detect obstacles. The five on-board lasers project stripes onto the ground (see Fig. 2.13), and four selected lines in the stereo cameras are scanned to build up a 20-point range "image" of the terrain immediately in front of the rover: the perturbation of a detected laser spot from its nominal position within a scanline indicates the change in elevation at that coordinate, relative to flat ground. This ephemeral terrain model, generated on-board during execution of the "Go To Waypoint" command, is used by the rover to perform hazard detection (dropoffs, steps, and steep slopes) and reactive, behaviour-based avoidance—

to Earth by the spacecraft, assess the state of the rover, and generate constraints to be considered in the next sol's operations. The team had to provide an initial evaluation of the state of the vehicle within approximately two hours of the end of the downlink communication session. The uplink team absorbed the downlink team's conclusions, then met with the project scientists to determine the course of action for the next sol, and to hash out conflicts between desired experiments and rover engineering requirements. Finally, the uplink engineers spent the next several (8-10) hours laboriously building, documenting, and very carefully reviewing the command sequences, using the Silicon Graphics Inventor®-based Rover Control Workstation (on an SGI Onyx 2).

Figure 2.13: Image of laser stripe from front left rover camera.
(courtesy JPL)

including the ability to verify that the vehicle's turning circle is obstacle-free, and to "thread the needle" between rocks in close quarters—on the way to the goal. The rover is also capable of detecting other types of hazards, such as contact (with bump sensors mounted on the solar panel and body), tipover (with accelerometers), and articulation (with potentiometers on key suspension link pivots). Rover odometry keeps track of a "virtual hazard": the rover's proximity to the lander.[‖] Operators can command whether the rover will, upon detecting a specific type of hazard, either abort the traverse, ignore the hazard, or avoid the hazard autonomously. Future rovers will potentially use a combination of full passive stereo imaging, contact sensors, accelerometers, potentiometers, encoders, sun sensors, turn rate sensors, and laser "pushbroom" sensing, to aid with navigation [80].

However, even in the most relaxed cases, Sojourner required navigational supervision from the ground. Since Sojourner's "Go To Waypoint" command has only been tested for short distances, the rover driver needs to specify subgoal positions in clear areas every few meters (the longest distance traversed by Sojourner during a single sol was 7.769 m on Sol 32 [41]). Even more importantly, the distance travelled by the rover each sol is limited by accumulated dead reckoning error, which is on the order of 5-10% of the distance travelled, and caused in part by wheel slippage (very sensitive to differences in substrate) [54]. The drift exhibited by the on-board turn rate sensor is even worse, at approximately 13° per sol, forcing the usage of wheel odometry to measure turns—a notoriously inaccurate method.[**] As a result,

[‖]This tunable parameter (3m default) defines the closest the rover may venture near the lander, in order to avoid fouling the wheels with the lander's deflated air bags (left over from landing) (see Fig. 2.6).

[**]It has been hypothesised that this drastic drift rate was due to a circuitry error, and

the Sojourner rover is highly dependent upon the rover drivers' ability to localise the rover in the lander-based coordinate system at the start of each sol. Thus, the rover is constrained to remain within 10-15 m of the lander, within the region of good stereo resolution from the IMP cameras. Moreover, since the rover is localised using generally a single "end-of-day" image, the results are sensitive to registration errors in lander imagery, further limiting safe traverse distances. Problems with the Earth-Mars and lander-rover communications links aggravate the issue: for example, if the end-of-day image is not received for a given sol, the rover cannot be localised, and desired traverses for the next sol must be cancelled. Unfortunately, problems with the DSN are common. Thus, future rovers must incorporate autonomous localisation techniques to overcome these difficulties.

Other limitations to Sojourner's autonomous navigation abilities, which should be considered in the design of future rovers (though not all are treated in this thesis), include the fact that the body-mounted cameras afford a low vantage point, preventing anticipation of more distant hazards. The 20-point terrain model is sparse, and could potentially either miss a hazardous obstacle or misclassify traverseable terrain. The heuristic path planner has no guarantee that the rover will reach the goal, nor stop if the goal is unreachable; since the goal is specified as a coordinate in the lander-based grid, dead reckoning error affects the rover's knowledge of her target's location. The terrain map, sparse as it is, is not retained for future reference. The Sojourner rover's slothful speed also contributed to navigation problems: the rover was unable to cover more than approximately 10 m per sol (since she must traverse during hours of peak sunlight), the slow speed exacerbated the gyro drift problem, and sometimes operators were simply unable to fit traverse and other rover ops (e.g., imaging) in the same sol. Sojourner was also sensitive to sensor failure: the accelerometers proved themselves unreliable, and needed to be disabled precisely when tipover information was needed most. Finally, the lack of extensive pre-landing testing, forced by time and budget constraints, caused the operators to act conservatively. Many of Sojourner's autonomous capabilities had

not intrinsic to the rate sensor itself.

not been tested in Marslike conditions, resulting in a general lack of confidence in the rover's ability to avoid hazardous configurations. Thus, it is imperative that in order to be truly useful, new autonomous motion planners must be proven on appropriate vehicles in realistic terrain. Still, despite her limitations, the Sojourner rover is the most autonomous spacecraft to fly to date.

2.5 The Next Decade on Mars

The successes of the Sojourner rover enthused the scientific community, and resulted in a new plan for a series of rover missions to be sent to Mars at every favourable launch opportunity (roughly every two years), beginning in 2001 through at least 2005.[††] Although the planned mission scenario has evolved somewhat over the past few months, many of the rover requirements have remained stable. The proposed 2003/2005 rover missions both feature a similar rover, an advance over the Sojourner model, known informally as the Athena rover. The 2001 mission, on the other hand, will fly a virtual duplicate of the Sojourner rover, named Marie Curie.[‡‡] Although there are conceptual plans in the pipeline for JPL rover missions to Mars in 2007 and 2009, these projects have not yet reached even a preliminary design stage.

As is par for the course for planetary missions in their earliest stages, the scope, scenario, and even vehicle configuration for the next three Mars rover missions have altered rapidly; at times, changing even weekly. We describe here the original scenario for the 2001 mission, as laid out in its Announcement of Opportunity, June 30, 1997, and in subsequent discussion of the details among the rover project design team [1],[52]. This initial mission plan was the original motivation for the work presented in this thesis. The basic mission architecture is as follows: The rover is a flight microrover-class vehicle, with Sojourner heritage, but roughly 1.5 times larger (about 100 cm x 75 cm x 45 cm) and weighing less than 40 kg. After the rover is

[††]There is also a Mars mission for the upcoming launch opportunity in December 1998/January 1999. However, this mission was developed before the rover concept was proven, and features a robotic arm but no mobile companion.

[‡‡]The Marie Curie rover is actually the refurbished flight spare of the Sojourner rover, used during the Pathfinder mission to test controversial sequences in the "sandbox" test environment before the sequences were sent to Mars.

deployed, after a soft (propelled and guided) landing, it will operate on the surface of Mars for up to one Earth year. During this time, the rover has a direct link to a companion orbiter, through which the rover communicates with Earth. The scenario calls for the rover to abandon the lander after exploring its environs, and traverse up to a kilometer between each of several approximately 100 m^2 regions of more detailed exploration, *in situ* analysis, and sample caching (see Fig.2.16). The 2001 rover, and a similar rover sent in 2003, will collect samples along the way and cache them on-board. A fast, light, sample return rover to be flown in 2005 will land near one of the two dead rovers, retrieve its cachebox, and load the box into an ascent vehicle to return the samples to Earth. There are two communication opportunities with the orbiter each sol, at 4AM and 4PM Mars local time (MLST, for Mars Local Solar Time). Note that these periods are outside the time when solar energy is available (roughly 8AM until 5PM MLST). In particular, the orbiter—and its link to Earth—is not available during rover traverses, which must be executed during times of peak solar availability, e.g., 10AM to 3PM MLST. Furthermore, the rover is expected to traverse much longer distances between communication opportunities with Earth: up to 50-100 m during a single sol (recall that Sojourner traversed roughly 100 m total during her entire mission), for a total traverse distance of up to 10 km for

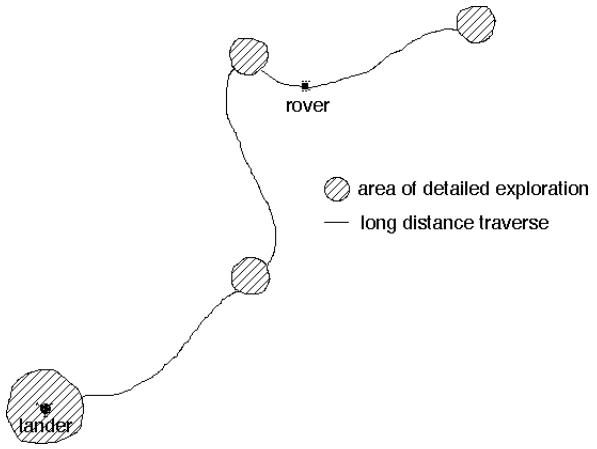


Figure 2.14: Sketch of original 2001 mission scenario. In this mission design, the rover leaves the unneeded lander far behind, ranging up to kilometers away. The areas for detailed exploration and sampling are roughly 100 m^2, and separated by up to a kilometer from similarly explored regions.

the year. Moreover, it will not be possible to anticipate (and thus pre-program) the proper reactions of the rover to the terrains encountered during the traverse—even if dead reckoning error rates allowed for lengthy open-loop traverses—due to the fact that the rover is working in unknown, rough terrain. (Recall that the resolution expected from Mars orbiters, for example, is roughly 300 meters/pixel, with only isolated "postage stamp" regions achieving the highest resolution of 1.4 m/pixel [50]. Orbiter camera pointing limitations prohibit attempting to use these highest-resolution images for rover navigation or localisation.) Thus, an efficient, on-board planner is needed to ensure the rover will make acceptable progress toward each goal, and to achieve each goal accurately, during long stretches of unsupervised time each sol.

In the summer of 1998, the plans for the first three Mars rover missions changed drastically. Due to various political and technical considerations, the ambitious science payload ("Athena") meant for the 2001 rover was shifted to 2003 and 2005. After a period of uncertainty regarding the 2001 mission, Congress decreed that Marie Curie, the Sojourner flight spare, would be flown. The remainder of this section describes the detailed scenarios for the 2001, 2003, and 2005 Mars rover missions as they stand at the time of this writing [53], [80].

All three missions will be carried to Mars on Viking-type soft landers, which in the case of 2001, requires the rover to be deployed by the lander's robotic arm. Also of note, many of the experiments meant for the "Athena" rover are now situated on the lander, including the PanCam (for Panoramic Camera) system. Once deployed, the rover navigation will be directed by the PanCam according to the same model used in commanding Sojourner. Further, Marie Curie will be communicating via RF modem to the lander, as did Sojourner. However, the direct-to-Earth link enjoyed by the Pathfinder lander will be replaced by an orbiter relay. In general, the orbiter will make two passes per sol, likely at 2:30AM and 2:30PM. The expected downlink telemetry per sol—including all lander and rover communications—is approximately 40 Mbits. Once deployed, Marie Curie will conduct similar operations as did Sojourner, performing soil and rock APXS analyses. However, the PanCam

has 3 times the spatial resolution of Pathfinder's IMP cameras, enabling Marie Curie's operators to extend her work space to at least 20 m from the lander. To facilitate navigational autonomy, LBAL (Lander-Based Autonomous Localisation) will be tested as a technology experiment. During these tests, Marie Curie will update her position up to every half-vehicle length of traversed distance, and typically every few vehicle lengths. Unfortunately, it is unlikely that operators would be able to generate rover command sequences each sol: due to latency in the satellite relay, sequence turnaround time could be only 6-8 hours.* More likely, Marie Curie will be commanded only every other sol. Thus, a reliable autonomous motion planner which satisfies all of Sojourner's previously-described constraints, in tandem with LBAL, would be of tremendous aid in increasing the useable length of time between new command sequences.

For the 2003 and 2005 missions, the larger Athena rover will be deployed down a ramp. The current 03/05 design calls for a wide-track rover roughly twice the size of Sojourner, about 1.45 m x 1.2 m x 0.6 m and weighing 70 kg. Her science payload will include the PanCam and a thermal emission spectrometer (Mini-TES) on a mast; a belly-mounted core drill; and four instruments (Mössbauer spectrometer, microimager, APXS, and Raman spectrometer) in the carousel end-effector of her 4 degree of freedom arm. There are two manipulation-support cameras mounted

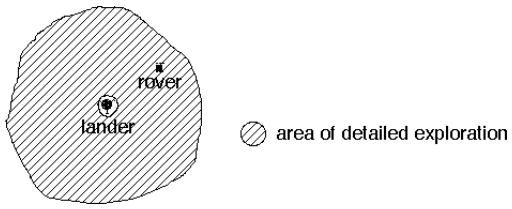


Figure 2.15: Sketch of updated 2001 mission scenario. The area of detailed exploration is an annulus centered around the lander, ranging from 1.5 m to at least 20 m from the lander.

*Satellite latency derives from the following conditions: Telemetry can only be downloaded from the orbiter several hours after the data is transmitted from the lander, in order to ensure a full data set. Then, the operators must transfer the finished command sequence to the orbiter 1 hour before its communications pass over the lander. Within the time remaining, the operators need to interpret telemetry, receive activity sequences from the science group, build the command sequence, and prepare the final sequence for uplink.

on the belly, though it is not clear whether these cameras will be a stereo pair. The rover will be powered by a solar panel, as well as by lithium ion rechargeable batteries which can store sufficient energy for one full sol's operations requirements. For navigation, the rover will carry 2 body-mounted stereo camera pairs, fore and aft, as well as the NavCam on the mast. The NavCam, with its 45° field of view (FOV), covers much more ground than the PanCam's 8° FOV; however, the PanCam yields much higher resolution images. She will also carry an optics-based sun sensor, accelerometers, a turn rate sensor, suspension-pivot potentiometers, and motor encoders. Of note is the fact that the lander will not have its own camera system, other than its descent imagery camera; thus all navigation and localisation must use rover imagery. Her computing environment consists of an R3000 running at 10 MIPS, with approximately 6 MB total memory. The Athena rover's surface operations include drilling for core samples, and performing analyses of rock and soil in immediate vicinity of the sample site. A primary requirement is acquiring an appropriate diversity of samples. Thus, given the amount of time (potentially several sols) required for a thorough analysis of a site due to the number of instruments on board, it is strongly desired to reduce the time used by repetitive operations, such as traversal.

The current mission architecture for 2003 is as follows: Before descending the ramp, the rover may take a 360° panorama of the immediate surroundings. The rover's surface mission consists of three sample-collection sorties, each riskier than the last. The first, "insurance," sortie directs the rover to the nearest site of scientific interest (chosen by scientists using PanCam, Mini-TES, and descent imagery[†]). The rover will collect a small number of samples to return to the lander's MAV (Mars Ascent Vehicle). The return of samples to the MAV will be a carefully choreographed series of steps, based upon both the basic complexity of transferring the sample cache to the MAV as well as planetary protection requirements. In particular, a key autonomous operation will be re-acquiring the ramp, then re-ascending to

[†]In the 2001 mission, Marie Curie will help provide ground truth (with her APXS) for sites chosen via PanCam/Mini-TES, to calibrate sample site selection for the 2003/05 missions.

reach the MAV. (While atop the lander, rails pre-align the rover with the ramp. However, it is possible that the rover could drive off the edge of the ramp if she is not properly aligned during ascent.) The second sortie will range further, and gather the primary mission sample set. The final sample set is part of the extended mission. Each of the sorties will take place within 1 km of the lander. The Athena rover will have a three-part cachebox, where each part of the box is transferred to the MAV upon the end of its respective sortie. Therefore, the rover will be unable to cache further samples after the end of the third sortie, opening the possibility for longer exploration traverses. Clearly, for any of the sorties/traverses, if the operators choose to travel beyond the range of the currently visible area, an on-board autonomous path planner is required. To round out the mission description, the baseline communications design is a direct-to-Earth link via the lander, although such a link may only be available for an hour or two per sol (including both uplink and downlink), due to power constraints. The expected sequence turnaround time is comparable to that for 2001. The duration of the 2003 mission is 90 sols for the primary mission (1st and 2nd sorties), plus a 90-sol extended mission. By comparison, this mission will last more than twice as long as Pathfinder, and by sol 83, the Pathfinder "rover drivers" were exhausted, even with 13-14 hour sequence turnaround times. Some of this effort may be ameliorated by commanding the rover only every other sol. Still, it is clear that an autonomous motion planner will

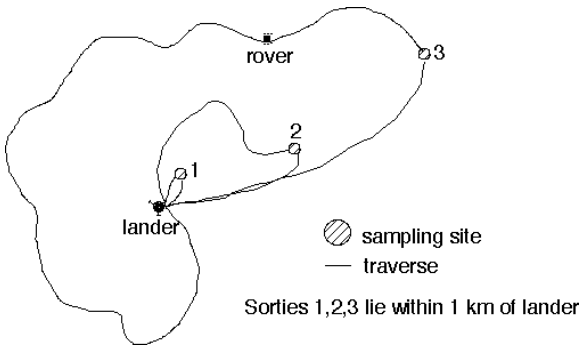


Figure 2.16: Sketch of updated 2003/05 mission scenarios. The rover executes three sorties for sample collection, each riskier than the last. All three sorties remain within 1 km of the lander. Not shown: in the 2003 mission, it is possible that the rover will conduct additional extended-range exploration after the third sortie.

significantly contribute to mission success.

The 2005 launch opportunity is the most difficult mission, since at that time Mars will be very near conjunction[‡]. For this mission, all of the issues from 2003 are exacerbated. The 2005 mission will be very challenging operationally: the spacecraft will be 2.5 AU's away (the farthest point from Earth in Mars' orbit), which means that the time needed for round trip communications will be greatest (approx 40 min). In addition, the available data volume per sol is drastically reduced, due to signal deterioration with distance, and power constraints. Budgetary and political factors induce this particular mission to be launched from Kourou, Guiana, on an Ariane 5, based upon an agreement with the French. Due to the particulars of the launch site, the spacecraft will arrive 82 days prior to conjunction. Therefore, the mission will have less than three months to be completed, and to launch the MAV, since the lander is not expected to survive the few months it will be unable to communicate with Earth during conjunction. The mission duration may be even more curtailed if it is decided to attempt to rendezvous the sample return spacecraft with the orbital caches and to begin the flight back to Earth before conjunction. If this option is chosen, the surface operations phase is constricted to roughly 30 sols. Thus, since the mission architecture is the same as for 2003, the 2005 rover will have only about a month to execute all three sorties and return the samples to the MAV. As a result, it is imperative that the rover be as autonomous as possible, particularly with respect to traversal.

A key aspect of the new scenarios, compared with the Pathfinder mission, is fewer communications opportunities, with the rover expected to accomplish more between opportunities. Specific to the 03/05 mission architectures, the rover will have to traverse longer distances—up to 100 m/sol, roving up to 1 km from the lander, as opposed to Sojourner's total of ∼102 m within 12.5 m of the lander—without aid from Earth.

[‡]Conjunction occurs when the subject planet lies directly on the opposite side of the Sun from Earth.

2.6 The Need for Autonomy

The plan for the next decade on Mars, then, requires rovers with a significantly higher level of autonomy than the Sojourner rover, particularly in the area of navigation. The rovers will be expected to traverse much longer distances, accurately, without supervision—up to as far in each sol as Sojourner traversed during her entire mission. The limited communications opportunities, coupled with the limited expected lifetime of surface spacecraft, impels us to enable the rover to accomplish as much as possible autonomously in order to maximise science return. In the area of long distance navigation, this means that the rover should be able to be given a high-level goal, such as a coordinate in the planetary or lander-based coordinate system, and not require detailed, intensive supervision from Earth-based "drivers"—either human or ground-based autonomous planners—to reach the goal. Thus, the need is clear for an autonomous, on-board path planner which satisfies the stringent flight and mission constraints.

A practical on-board path planner must be *sensor-based*; that is, the planner must not assume prior knowledge of the area. As noted earlier, orbiter data can be expected to have a resolution of generally only about 300 meters/pixel for any given region. It might be useful to incorporate knowledge gained from descent imagery, but even in this case, data with sufficient resolution for navigation could only be expected for regions very close to the landing site. Indeed, the limited scope of the rover's knowledge of its environs must be factored specifically into the process of path planning. Due to the limited range of the rover's sensors, the planner must build a path incrementally from the rover's current position to the goal. The approach is rather like having a moving window centered on the robot which illuminates only the immediate surroundings; from this limited knowledge the best choice for progressing toward the goal is determined.

Further, the generated path should be *locally optimal*. That is, each segment of the path is optimal (as measured by path length), when only the obstacles able to be sensed by the robot along that segment are considered. As the range and angular coverage of the robot sensors approach infinity and 360 degrees, respectively,

the locally optimal path approaches the globally optimal solution in many cases, particularly if the rover sensors are able to see over the surrounding rover mobility obstacles (e.g., untraversable rocks).

In view of the terrific constraints upon computational effort, memory, and time, the planner should also minimise the need to acquire data solely for purposes of motion planning. That is, the planner should be able to use the available sensor array in an efficient manner to obtain—with a minimal number of sensor queries—the information needed to preserve the planner's useful properties and to progress toward the goal.

Finally, the rover should be guaranteed to reach the goal, or to be able to discover that the goal is unreachable and therefore halt. This property, *completeness*, along with local optimality, ensures that the rover acquires the goal—or recognises that the goal is unattainable—in reasonable time. Thus, the rover demonstrates predictable behaviour, and in the end, allows more time for science operations. A companion property, *correctness*, ensures that the generated path is obstacle-free.

2.7 The Path to the Future

In response to these needs, we have developed a practical autonomous path planner, tuned for the flight microrovers designed for the next decade of Mars missions. The planner, "RoverBug," is sensor-based, locally-optimal, complete and correct. It also satisfies the hard constraints of this class of flight rovers, including minimising memory usage, computation, and eliminating excess rover motion and sensing. This planner significantly augments microrovers' autonomous navigation ability, which in turn will aid in producing successful mobile robot missions. And thus will our ability to discover more about those alien worlds be extended, perhaps further than could be possible even by venturing out into the solar system ourselves. For, as stated by James Killian, Eisenhower's science advisor, in 1961, "the really exciting discoveries in space can be realised better by instruments than by man" [28].

Chapter 3

Upon the Shoulders...

3.1 Introduction

In any discussion of motion planning, and particularly during the algorithm development which follows in Chapter 4, there are certain basic concepts which should be reviewed in the interest of clarity. We therefore briefly address here such topics as *configuration space* and *freespace*. In addition, in Section 3.3 we include a more detailed discussion of the TangentBug algorithm developed by Kamon, Rivlin, and Rimon [21], [23], since the Wedgebug algorithm presented in this thesis is inspired chiefly by this work and bears several similarities. Finally, Section 3.4 treats a result from topology and stratified Morse theory, used in the proofs of completeness for both Wedgebug and TangentBug. We use this result to correct a minor flaw in the proof presented in [21], and then extend the proof's domain of applicability.

3.2 Concepts from Motion Planning

In his book, *Robot Motion Planning* [36], Latombe succinctly describes the notion (originally developed by Lozano-Pérez [39]) of a *configuration space*, $\mathcal{C}$. The underlying concept is to represent the real-world robot as a point in an appropriate space, and to map obstacles into this same space. Then, the space contains a concise representation of the robot's geometrical constraints on motion, and a motion planner needs only to consider the path of the single point which represents the robot.

The *configuration* q of an object A specifies the exact position and orientation of A relative to a fixed reference frame. Therefore, the *configuration space* (often referred to as "C-space") of A is the set of all possible configurations of A. For a robot with n degrees of freedom (DOF), its C-space, $\mathcal{C}$, is an n-dimensional space (typically an n-dimensional manifold), since there is a one-to-one, onto map between q and a specific arrangement of the robot's joints and the robot's position within its (real world) workspace.

Obstacles are mapped into C-space by determining which configurations of the robot produce collisions with an obstacle; these configurations are deemed forbidden. Let $A(q)$ denote the location of A's particles when A is in configuration q. A C-space obstacle (or "C-obstacle") associated with a physical obstacle B is defined as

$$\mathcal{C}B = \{q \in \mathcal{C} \mid A(q) \cap B \neq \emptyset\}.$$

The complement of the C-obstacles is termed the "freespace," $\mathfrak{F}$:

$$\mathfrak{F} = \mathcal{C} \setminus \cup_i \mathcal{C}B_i.$$

Motion plans are constructed in $\mathfrak{F}$. Note that the closure of freespace includes the obstacle boundaries. Since generally a robot is allowed to touch an obstacle (but not penetrate it), the word "freespace" (and symbol $\mathfrak{F}$) will often be used to refer to the closure of freespace as well. In the case illustrated in Figure 3.1 (part (a)), the robot is a rigid body in a 2D environment (its *workspace*), and it is capable of translating in any direction (i.e., horizontally and vertically) but not rotating. Thus, this robot has 2 DOF, and its configuration space (shown in (b)) is equivalent to $\mathbb{R}^2$.

Configuration space is a useful concept for classical motion planners aimed at creating paths for (holonomic*) multi-DOF robots. Classical motion planning is a

*Holonomic robots are capable of motion directly between configurations: for example, a robot which can move forward, backward, and side-to-side freely in an obstacle-free 2D environment. An automobile is an example of a non-holonomic vehicle; other techniques must be used (perhaps in addition to C-space) to capture the inability of a car to move

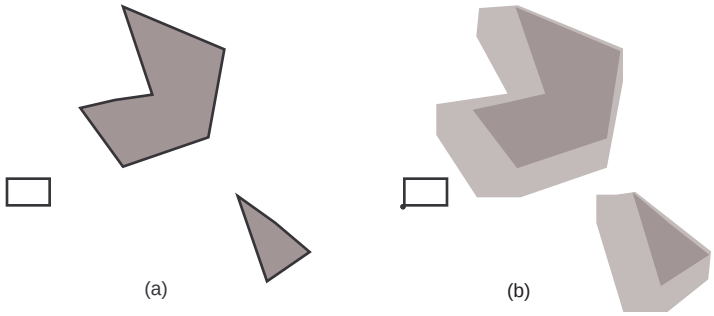


Figure 3.1: A sketch of the configuration space of an oblong robot with the two polygonal obstacles shown in (a). The robot can slide in any direction, but not rotate. The polygons in (b) are the configuration space obstacles corresponding to the point highlighted on the robot, which would then be used by a motion planner to plot the robot's trajectory.

two-step process. In the first step, the C-obstacles are computed from knowledge of the physical obstacles and of robot geometry. The second step constructs a path within $\mathfrak{F}$. The resultant path can then be translated back into actual robot configurations through an interface with the robot's automatic controllers. As a consequence of the power of this concept, many of the motion planners discussed in this thesis are concerned exclusively with motion within C-space.

3.3 TangentBug

We now turn our attention to the TangentBug algorithm developed in 1995 by Ishay Kamon, Elon Rimon, and Ehud Rivlin at the Technion in Israel. The most detailed description of this algorithm is contained within their 1995 technical report [21]. We present here a more compact treatment, highlighting those aspects of the algorithm which will come into play in Chapter 4.

The thrust behind TangentBug is to find a way, using solely the robot's on-board sensors, to navigate though previously unknown (2D) terrain, avoiding collision with all encountered obstacles (*correctness*) while ensuring that the robot will either achieve the goal or stop if the goal is unreachable, both after traversing only a finite distance (*completeness*). As discussed in Chapter 1, several algorithms—

directly sideways.

most notably the Bug algorithms developed by Lumelsky and Stepanov in 1987 [40]—accomplish this goal. However, many of these other algorithms (including Lumelsky's Bug algorithms) can produce quite lengthy paths. TangentBug was developed as a new member of the "Bug" family—an algorithm "which combines local planning with global information that guarantees convergence" [21]—which specifically incorporates range information to produce *locally optimal* paths. In other words, TangentBug determines the shortest path possible, given the fact that the robot has knowledge only of obstacles within range of its sensors (which may have limited range). (It should be noted that none of this prior work has dealt with gaze control issues for robots whose sensors have limited angular scope.)

TangentBug makes heavy use of a construct called the *local tangent graph*, or LTG. First, we briefly review the global Tangent Graph (also commonly referred to as the "reduced visibility graph" [36]). Given a 2D configuration space populated with polygonal[†] obstacles about which we have full prior knowledge, and given an initial configuration q_{init} and a goal q_{goal}, the Tangent Graph is an undirected graph whose nodes consist of q_{init}, q_{goal}, and all of the obstacles' convex vertices[‡]. The graph edges, in turn, comprise those line segments which connect pairs of nodes in such a way that the entire segment lies within $\mathfrak{F}$, and if the segment is extended into a line, that line is tangent[§] to obstacles containing the segment's endpoints (see

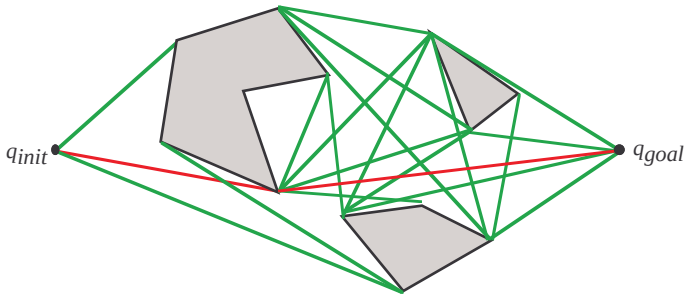


Figure 3.2: The Tangent Graph.

[†]The Tangent Graph can also be constructed for curved 2D obstacles, as detailed in [38].

[‡]i.e., a vertex v of an obstacle O such that in a neighborhood $N \in O$ of v, the segment $\{x | x = \lambda x_1 + (1-\lambda)x_2;\ 0 \leq \lambda \leq 1\} \in O,\ \forall x_1, x_2 \in N$.

[§]A line l is *tangent* to a C-obstacle CB at a vertex x iff in a neighborhood of x, the

Figure 3.2). It can be proven that the shortest possible path that connects q_{init} to q_{goal} through $\mathfrak{F}$ must be included within the Tangent Graph [67]. So, a planner need only search this graph to find the globally optimal path (measured by path length).

However, if the planner does not already have full prior knowledge of its environment, the robot must use its sensors to determine a clear path to the goal. In this case, Kamon, Rimon, and Rivlin assume that the robot has an array of sensors which can return range data within a distance R from the robot, covering a 360° circular arc. The sensors are "sonar-like" in that they return the range to the closest obstacle (within the sensor's range limit) in a given direction, yielding the effect that the sensors are "blocked" by obstacles and cannot "see" beyond the closest obstacle boundary. (See Figure 3.3.) (An alternate perspective is that the obstacles are "wall-like;" that is, obstacles block both motion and sensing.) Since now the robot can detect obstacle boundaries only as disembodied curves (terminated on both ends by discontinuities in range measurement), we model the sensed obstacles as thin walls along those curves. As an aside, the correlation between sensed obstacles and actual obstacles is not one-to-one, as illustrated in Fig. 3.3. In part (b) of the figure, for example, obstacles O_1 and O_2 correspond to the same physical obstacle. However, the robot's limited sensor range, coupled with sensor

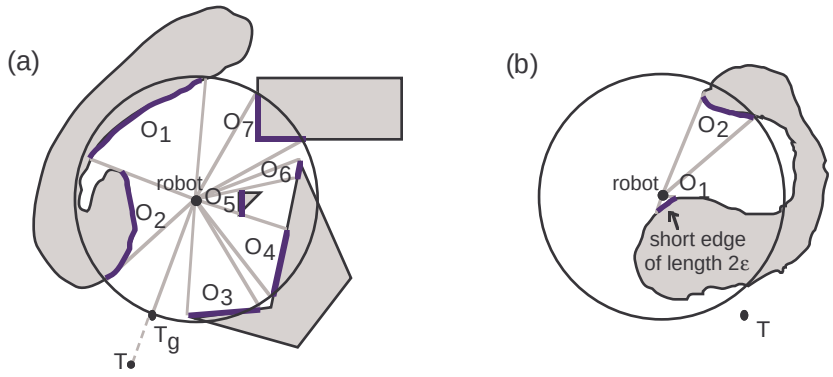


Figure 3.3: The LTG, assuming sensors with range R and omnidirectional view. The sensed obstacles O_i are modelled as thin walls. (Note that there is not a one-to-one correspondence between sensed- and actual-obstacles.) The LTG, with edges radiating out from the robot's position, connects the robot with the endpoints of the O_i and with T_g.

interior of CB lies entirely on one side of l [36].

occlusion, prevents the robot from recognising that the two segments are parts of a single connected boundary. A further assumption states that the robot is never precisely on an obstacle boundary, but instead a slight distance away, so the sensed boundary of the "contacted" obstacle does not degenerate into a point. Instead, the sensed boundary of a contacted obstacle is represented by a short line segment with length 2ϵ, tangent to the actual obstacle's boundary and centered at the "contact point" (Figure 3.3(b)).

Further, Kamon, Rimon, and Rivlin assume that the robot itself is a point, capable of movement in any direction, and that it is rotation-invariant. Then, the range information used by the planner is precisely the information returned by the robot's sensors: the distance to the closest obstacle in a given direction. Our start and target points are now positions in 2D, rather than more complex configurations; we will call these points S and T, respectively. We retain the notion of freespace as the set of all allowable positions.

Now we may define the local tangent graph (LTG), at a position x (Figure 3.3). The LTG is essentially the Tangent Graph restricted to the visible region at x, bounded by the set of sensed obstacles $\{O_i\}$ and by $B_x(R)$, the ball at x with radius R. As such, the LTG's nodes consist of all sensed obstacle vertices, including the sensed obstacles' endpoints, as well as x (as the robot's current starting point). T is included only if $\mathrm{d}(x,T) \leq R$, where $\mathrm{d}(a,b)$ is the Euclidean distance between a and b. Otherwise, in order to include the influence of the goal, we optionally add a new node, T_g, at the intersection of the circle at x with radius R ($\partial B_x(R)$) with the line segment $\overline{xT}$. T_g is added if and only if the portion of the line segment $\overline{xT}$ within $B_x(R)$ lies wholly in $\mathfrak{F}$, and $\mathrm{d}(x,T) > R$. T_g is essentially the projection of the goal onto the visible region. The LTG's edges comprise all line segments in $\mathfrak{F}$ between x and the sensed obstacle endpoints (and T_g, if it has been added). Although strictly speaking, those edges in $\mathfrak{F}$ which connect two obstacle vertices (or an obstacle vertex with T_g) are part of the LTG, they are never used by TangentBug for motion planning, and so will never be explicitly constructed.

3.3.1 Overview of the TangentBug Algorithm

Like the earlier members of the "Bug" family, TangentBug features two basic motion behaviours, "motion-to-goal" and "boundary following," which together with the behaviour-switching conditions guarantee convergence to the goal. During motion-to-goal, the distance from the robot to T decreases monotonically. Boundary following, on the other hand, attempts to escape a local minimum in the function $\mathrm{d}(\cdot, T)$. The robot constantly senses its environment and updates the LTG accordingly; in turn, the LTG is used by the active behaviour mode to determine the robot's next motion.

The algorithm begins with the motion-to-goal mode, during which the planner searches the LTG for the *locally optimal direction*, which is the direction along the shortest path to the goal according to the robot's limited field of view (FOV). As the robot moves, motion-to-goal continues until either the robot reaches T, or the planner detects that motion along the locally optimal direction will trap the robot in a local minimum of $\mathrm{d}(\cdot, T)$, caused by an obstacle between the robot and the goal. When the latter situation arises, the robot switches to boundary following. This mode first chooses a direction to follow the obstacle boundary in order to escape the local minimum. During boundary following, the planner continues to update the LTG, and monitors the graph for satisfaction of the *leaving condition*, which indicates that progress toward the goal can be made along a path which brings the robot closer to T than at any point traversed so far, and therefore the robot may leave the obstacle boundary. The planner also checks the LTG for shortcuts which can be taken as the robot follows the obstacle boundary. The boundary following behaviour ends either when the leaving condition is met, in which case the robot returns to motion-to-goal, or when the target is reached, or finally if the robot detects a loop—that is, the robot has circumnavigated the entire obstacle. If the robot cannot reach T nor meet the leaving condition before circumnavigating the obstacle, T must be hidden within the obstacle and unreachable; the robot stops and the algorithm halts. To summarise:

1. *Motion-to-Goal:* Move along the locally optimal direction towards T, until one

of the following occurs:

- The target is **reached**. Stop.
- The planner detects that motion along the locally optimal direction will trap the robot in a local minimum of $\mathrm{d}(\cdot, T)$. Go to step 2.

2. *Boundary Following:* Choose a direction for motion around the obstacle. Move around the obstacle boundary, using the LTG to determine shortcuts, while recording d_{reach}, the closest distance to the goal encountered so far along the obstacle boundary, until one of the following occurs:

   - The target is **reached**. Stop.
   - The planner detects that the leaving condition holds. Go to step 1.
   - The robot detects a loop: the target is **unreachable**. Stop.

We describe the motion-to-goal and boundary following behaviours in more detail below.

3.3.2 Motion-to-Goal

Recall that the basic function of the motion-to-goal procedure is to bring the robot monotonically closer to the target. This behaviour does so by searching a subgraph of the LTG, defined as $G1 = \{V \in \mathrm{LTG} \,|\, \mathrm{d}(V,T) \leq \min(\mathrm{d}(x,T), \mathrm{d}_{\mathrm{LEAVE}})\}$, where x is the robot's current position, and the parameter $\mathrm{d}_{\mathrm{LEAVE}}$ stores the farthest distance the robot may stray from T during motion-to-goal movement. Essentially, $\mathrm{d}_{\mathrm{LEAVE}}$, which is initially set at $\mathrm{d}_{\mathrm{LEAVE}} = \mathrm{d}(S,T)$ and is reset after each boundary following step, constricts the guaranteed distance from the robot to the goal during motion-to-goal to a series of ever-shrinking circles. The planner augments $G1$ by assigning to each edge (V,T) the length of the shortest path in $\mathfrak{F}$ (considering only the currently visible obstacles) between V and a new node at T, for each $V \in G1$. The algorithm then searches the augmented $G1$ for the shortest path to the goal, and the robot moves along the direction indicated by this path.

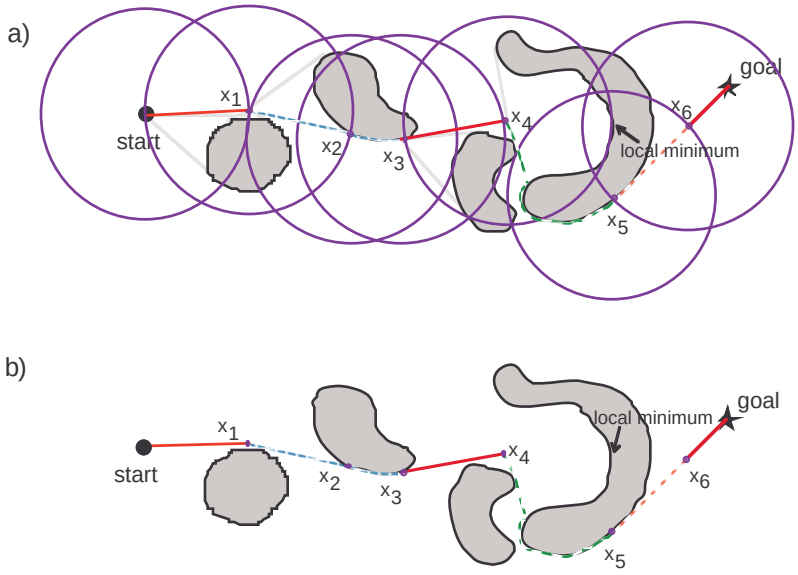


Figure 3.4: A sample execution of TangentBug. (a) Path generated by TangentBug, showing selected 360° views (with radius R) and associated LTGs along the way. Note that near x_1, the robot has detected an obstacle in its path, and must slide around it, maintaining progress toward the goal. At x_4, the algorithm switches to boundary following, until the leaving condition is met and motion-to-goal resumes at x_5. "Direct" motion-to-goal segments are solid and dotted, "sliding" motion-to-goal segments are dashed (x_1 through x_3), and boundary following segments are dot/dash (x_4 through x_5). (b) The completed path. (Compare with Figure 4.14 in Chapter 4.)

If the goal is achieved during this behaviour, the algorithm ends. Otherwise, the motion-to-goal behaviour continues until the robot detects that it is trapped within the basin of attraction of a local minimum in the distance to T, caused by an obstacle blocking the robot's progress toward the goal (the *blocking obstacle*). Such a local minimum is indicated when $G1$ becomes empty. At this point, the planner switches to its boundary following behaviour.

3.3.3 Boundary Following

The underlying idea of the boundary following mode is to circumnavigate the blocking obstacle, thus moving away from the basin of attraction of the detected local minimum in $\mathrm{d}(\cdot, T)$, until either progress toward the goal may be resumed or the

target is determined to be unreachable. Upon initiating a new boundary following step, the planner chooses a direction to follow the obstacle boundary by searching the entire LTG at x, augmented in a similar fashion as described above, to find the shortest path to T. The robot also records d_{reach}, the closest distance to T from any visible point on the obstacle boundary. Unless the robot happens across the goal, in which case the robot moves to T and the algorithm ends, the robot continues around the obstacle boundary, recording the minimum distance to the goal encountered so far along the obstacle boundary (d_{reach}) and updating the LTG, until either the leaving condition is met or the robot detects that it has completed a loop around the obstacle. The leaving condition holds when $\exists V \in \mathrm{LTG}$ such that $\mathrm{d}(V,T) < \mathrm{d}_{reach}$. That is, there is a node in the LTG which lies closer to T than any point so far encountered on the obstacle boundary, and thus the robot can leave the obstacle boundary and resume its motion toward the goal. Indeed, at this point the algorithm returns to its motion-to-goal behaviour, first setting $\mathrm{d}_{\mathrm{LEAVE}} = \mathrm{d}(V,T)$. If, on the other hand, the robot completes a loop without satisfying the leaving condition, then as previously discussed, T is deemed unreachable, and the algorithm halts.

3.3.4 Proof of Convergence

We will not give a detailed proof of the completeness of TangentBug, since such a proof can be found in [21].[¶] Rather, here we present selected relevant results (the first of which actually appears in [22], a technical report covering a 3D version of TangentBug called 3DBug), as well as a brief overview of the completeness proof. Some of the results which appear here will be used in the sequel.

The basic idea of the proof of completeness for TangentBug is that each motion segment can be classified as a particular type. Each motion segment type in turn can be shown to have finite length, and finally it is shown that there are a finite number of each type. Thus, the algorithm halts after the robot has traversed a finite

[¶]We note, however, that there is an error in the proof for TangentBug using range sensors in [21], in the proof for Lemma 4.10 in that paper. A corrected version of the proof, changed slightly for the Wedgebug algorithm, appears in Chapter 4.

distance. The proof is clinched when it is shown that the robot will achieve T, if the goal is reachable from the robot's initial position. Proposition 2.1, presented below, is used to help classify the types of motion segments, since it shows that we need only consider segments which point directly towards the goal or which skirt the blocking obstacle. (The proposition will also be used in Chapter 4 to justify the choices of where to point the limited sensing array for the Wedgebug algorithm.)

Proposition 2.1 (from [22])**.** *Consider a planar polygonal environment, with a blocking obstacle between the robot location x and the target T. Then the* **shortest path** *from x to T,* **considering only the visible obstacles**, *must pass through an endpoint of the blocking obstacle.*

Proof. The sketch of the proof is as follows: Let γ_1 be the shortest path from x to T which passes through the right endpoint of the blocking obstacle. Let γ_2 be the shortest of all possible paths which circumvent the blocking obstacle on the right, considering only the visible (thin wall) obstacles (see Fig. 3.5). Construct polygons $Poly_1, Poly_2$ from the edges $\overline{xT}$ and γ_1 and γ_2, respectively. Then, the polygons are convex, since it is known that shortest paths must pass through convex obstacle vertices, and $Poly_1$ is included in $Poly_2$, since it can be shown that γ_2 does not cross γ_1 (though the two paths may partially overlap). Finally, it can be shown that a set of disjoint path segments $\overline{y_j y_{j+1}}$ along γ_2, constructed from the line segments $\overline{z_i z_{i+1}}$ of γ_1 in such a way that for each i and corresponding j, $\|\overline{y_j y_{j+1}}\| \geq \|\overline{z_i z_{i+1}}\|$, does not cover γ_2. Thus, γ_2 must be longer than γ_1. A similar argument for the left side shows that the shortest possible path from x to T must pass through an endpoint of the blocking obstacle. □

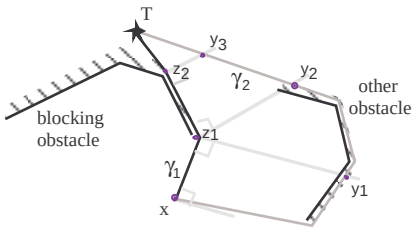


Figure 3.5: Illustration of γ_1, γ_2.

This proof can be extended to the case of non-polygonal obstacles—specifically, obstacles with continuous, *rectifiable*[‖] boundaries (which may still contain kinks):

[‖]Let $x(t)$, $a \leq t \leq b$, be an allowable representation of an arc C with initial point

Proposition 1. *Consider a planar environment populated by obstacles with continuous, rectifiable boundaries, with a blocking obstacle between the robot location x and the target T. Then the shortest path from x to T, considering only the visible obstacles, must pass through an endpoint of the blocking obstacle.*

Proof. The proof follows along the same lines as the proof for Prop. 2.1. Note that the generalised "polygons" constructed from $\overline{xT}$ and γ_1 or γ_2 are still convex, with $Poly_1$ included in $Poly_2$. Since the obstacle boundaries are rectifiable, so are the boundaries of $Poly_1$ and $Poly_2$. Finally, let Z be a set of chords along γ_1 such that the sum of the lengths of the chords in Z, $l(Z)$, equals $s + \varepsilon$, where s is the arclength of γ_1, and such that each straight-line subset of γ_1 is in Z. Construct a set of disjoint line segments $\overline{y_j y_{j+1}}$ along γ_2 from the line segments $\overline{z_i z_{i+1}}$ of γ_1 in the same manner as above, and note that again, this set—consisting of chords of γ_2—does not cover γ_2 and that the arclength of $(y_j, y_{j+1}) \geq \|\overline{y_j y_{j+1}}\| \geq \|\overline{z_i z_{i+1}}\|$ for each i and corresponding j. If we increase the number of nodes in Z such that $\varepsilon \to 0$, then $l(Z) \to s$, and $\|\overline{z_i z_{i+1}}\| \to$ the arclength of (z_i, z_{i+1}). So, we have the arclength of $(y_j, y_{j+1}) \geq \|\overline{z_i z_{i+1}}\| \to$ the arclength of (z_i, z_{i+1}) for each i and corresponding j. Thus, γ_2 must be longer than γ_1. □

After proving that the three types of motion segments: "direct" motion-to-goal (i.e., those segments which drive the robot directly towards the goal), "sliding" motion-to-goal (those segments which skirt a blocking obstacle while maintaining

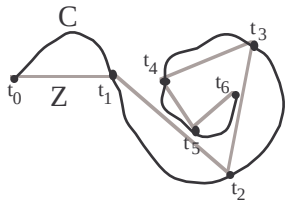


Figure 3.6: Illustration of a rectifiable curve.

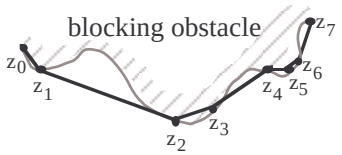


Figure 3.7: Sketch of the set Z associated with γ_1.

$x(a) = A$ and terminal point $x(b) = B$. Denote by $l(Z)$ the sum of the lengths of a set Z of chords whose endpoints are the points $\{t_\nu\}_0^n, t_0 = a, t_n = b$ (see Fig. 3.7). As $n \to \infty$ such that $\sigma(Z) = \max(t_\nu - t_{\nu-1}) \to 0$, if $l(Z) \to s < \infty$, then C is *rectifiable* and s is the arclength of C [31].

monotonic progress toward the goal), and boundary following segments each have finite length (using the reasonable assumption that obstacles have boundaries with finite length), it is shown that an entire motion-to-goal segment (consisting perhaps of several "direct" and "sliding" components) has finite length. Further, it is shown that the distance to the goal between successive local minima (along the robot path) is strictly decreasing**. Theorem 1 from [21] shows that the algorithm halts after finite path length. The theorem (along with Theorem 2, presented later) makes heavy use of the fact that there are a finite number of local minima of $\mathrm{d}(\cdot, T)$ which could be encountered by the robot. This statement requires a somewhat involved proof (as well as some restrictions on the environment), which will be given in Section 3.4.††

Theorem 1 (from [21])**.** *The algorithm terminates after a finite path.*

Proof. TangentBug switches from its motion-to-goal behaviour to boundary following only at a local minimum in $\mathrm{d}(\cdot, T)$. Further, it was shown that the distance to T between successive minima decreases, so the algorithm can switch to boundary following only once in each minimum. By Lemma 4.1 (see Section 3.4), there are a finite number of local minima of $\mathrm{d}(\cdot, T)$ within $\mathfrak{F}$. As a consequence, there are a finite number of motion segments. Since each segment has been shown to have finite length, the entire path has finite length. □

Finally, Theorem 2 from [21] demonstrates that TangentBug is complete.

Theorem 2 (from [21])**.** *If the target T is reachable from the starting point S then the robot will reach it in a finite path.*

Proof. It can be shown that if T is reachable from S, then every boundary following segment will terminate at a point where the leaving condition is satisfied, after finite path length. Therefore, since motion-to-goal switches to boundary following

**A proof for the similar result for Wedgebug is given in Chapter 4.

††The proof of this statement given in [21] was somewhat informal, and invokes the property that "any set of isolated points in a compact space...is finite," which is not strictly true. The treatment in Sect. 3.4 discusses and corrects this error, clarifies assumptions, and then extends the result to more general environments.

at a local minimum in the distance to T, and this switch can happen only once per minimum, it remains to be shown that there is a **last** motion-to-goal segment, which ends at T (assuming a generic environment). Motion-to-goal segments terminate either at a local minimum, or at the target. There are a finite number of local minima (Lemma 4.1), say N. Thus, if after visiting $N-1$ local minima the robot has still not reached T, the robot must encounter the N^{th} minimum. The boundary following segment thus begun must end with the leaving condition satisfied, since T is reachable. So, the robot begins the $N+1^{th}$ motion-to-goal segment. Since there are no more minima to terminate the segment, it must end at T. □

3.3.5 Trouble from Mars

The TangentBug algorithm we just described is a promising starting point for developing a motion planner for use by rovers on Mars. The algorithm is able to use local information gathered by the robot's sensors, augmented by just enough knowledge of the robot's position to track the target and to detect when the robot has executed a loop, to guarantee a locally optimal path to the goal (assuming T is reachable). At the same time, since the algorithm does not (necessarily) build a global model of its environment, and since it models obstacles as thin curves sensed by the robot, it does not require excessive memory to construct or store information about its environs.

However, TangentBug still has shortcomings when applied to the "rover problem" of navigation on Mars within the constraints discussed in Chapter 2, since some of its assumptions do not apply to this domain. Besides the clear examples—TangentBug assumes that the rover is a point robot capable of omnidirectional motion, for example, and it relies upon ideal dead reckoning for the global information it does require—there are many perhaps less obvious issues. A few examples follow: TangentBug assumes a continuously updated, 360° view of the environment. Not only is continual sensing a potential drain on available memory and computational capacity (since the sensed information must be processed to segment out obstacles), but the current designs for flight-like planetary rovers do not include

omnidirectional panoramic cameras[‡‡] (or other omnidirectional sensors). Thus, the available sensing array would need to be repositioned (or "panned") many times to acquire the full view, costing memory, computational effort, and time. Hence, any practical algorithm must attempt to minimise the amount of sensing via an automatic form of *gaze control*. Another issue is the fact that TangentBug assumes that the visible region is star-shaped, bounded by the sensed obstacles (or by the sensor range where there are no obstacles), whereas on Mars it is likely that the rover will be able to "see over" many of the obstacles it encounters. In addition, although the locally optimal nature of the resulting paths is certainly desirable, the planner does not include a notion of maximising safety while skirting around an obstacle. Finally, the planner does not include a mechanism for handling terrain roughness, nor for coping with alternate possibilities for goal designation (rather than as a simple coordinate point; some alternatives include driving with a certain heading, until a given distance or time limit, or angling toward a particular terrain feature).

Many of these issues are addressed in the following chapters. In particular, the problem of sensing with a limited field of view is specifically pursued in Chapter 4, and leads to the development of the Wedgebug algorithm. Several of the other areas are incorporated into the "RoverBug" implementation (a variant of Wedgebug tuned for an actual flightlike prototype rover) described in Chapter 5. A few of these issues are still open research areas, such as a practical method to incorporate terrain roughness in a more satisfying manner than is described in Chapter 5, or to utilise alternate goal designations. A discussion of the remaining open issues appears in Chapter 6.

[‡‡]Several groups, both at NASA and in private industry, have been developing such cameras, which use conical-section mirrors to image 360° at once, for use on rover-like vehicles. However, currently none have been flight-qualified, and it is extremely unlikely that such a camera would be added to the 2005 rover.

3.4 The Lynchpin

The proof of completeness for TangentBug (and later Wedgebug) hinges on the statement that there are a finite number of local minima* of $d(\cdot, T)$ which could be encountered by the robot executing the algorithm. This statement was given as Lemma 4.1 in [21] by Kamon, Rimon, and Rivlin.† The proof sketch provided in [21] assumes that the C-space is bounded. From this fact, it is stated, it follows that there are a finite number of obstacles, and that their boundaries have finite length. The crux of the proof relies on the fact that "any set of isolated points in a compact space...is finite" [21]. Next, Kamon, Rimon, and Rivlin state that the critical points of $d(\cdot, T)$ lie on the obstacle boundaries, which are assumed to be smooth, and note that $d(\cdot, T)$ restricted to a boundary is smooth. Recalling the definition of a Morse function as a smooth function, all of whose critical points are isolated, and that almost every smooth function on a smooth manifold is Morse, they conclude that there are finitely many critical points of $d(\cdot, T)$ on almost any obstacle boundary. Finally, they characterise the boundaries on which $d(\cdot, T)$ is not Morse—those which have T at the center of curvature at a critical point—and thus add the assumption that the environment must be generic, to avoid this situation.

However, the key statement in their informal proof, that "any set of isolated points in a compact set is finite," does not strictly apply in this situation. In addition, the lemma's assumptions are not clearly delineated, and the result is restricted to generic, bounded environments populated by (a finite number of) obstacles with smooth boundaries. Here we present a corrected, more detailed proof. Next, we extend the results first to obstacles with piecewise smooth boundaries, then to obstacles with piecewise C^1 boundaries in a not necessarily generic environment. (In all cases, we assume that the boundaries are continuous and rectifiable.) We assume that the reader has basic knowledge of real analysis, differential geometry, and of topology; good references for these topics are [69], [68], [29] (for real analysis), [31]

*or rather basins of attraction, as will be shown later

†Specifically, Kamon, Rimon, and Rivlin state that there are a finite number of local minima in the (assumed bounded) free configuration space.

(for differential geometry) , [18] and [19] (for topology).

A brief "roadmap" of the remainder of this section follows: After establishing the basic properties of the relevant space, we prove the proposition that all of the local minima of $\mathrm{d}(\cdot, T)$ encountered by the robot lie within a closed disc centred at T with radius $\mathrm{d}(S, T)$. Thus, we eliminate the necessity of assuming that the environment is bounded. Next, we establish the fact that a set of isolated critical points of a C^1 function on a compact set is finite; thus, we need only show that the critical points of $\mathrm{d}(\cdot, T)$ are isolated on the C-space (and that the gradient is continuous) to prove the result. Noting that the critical points must lie on the obstacle boundaries, we invoke stratified Morse theory to complete the proof. Next, we extend the result to piecewise smooth boundaries (with a finite number of pieces) by treating each piece as a distinct compact smooth manifold. Finally, we define the notion of a "generalised critical point," and prove an extended lemma for the case of obstacles with piecewise C^1 boundaries (again, with a finite number of pieces) in a not necessarily generic arrangement, replacing "local minima" with the concept of "basins of attraction." To finish the section, we demonstrate that Tangent Bug is complete, replacing Lemma 4.1 from [21] with our extended result.

3.4.1 Proof of Initial Result

First, we discuss the nature of the relevant C-space, and of $\mathfrak{F}$, the set of interest. We note that here (since we are concerned only with translations and not rotations in robot motion), C-space is a *metric space*; that is, it is a pair (X, ρ) of a set X and a metric ρ, such that ρ satisfies the standard properties of a distance function. In our case, C-space is embedded in $\mathbb{R}^2$, inheriting the standard Euclidean metric. Indeed, C-space is a *2-manifold*, locally topologically equivalent[‡] to $\mathbb{R}^2$.

We also note that the relevant portion of C-space (i.e., the portion of C-space containing local minima where the algorithm may switch modes) is closed and

[‡]More correctly, an n-manifold X is *locally diffeomorphic* to $\mathbb{R}^n$, which means that every point $x \in X$ has a neighborhood which is *diffeomorphic* to an open set $U \in \mathbb{R}^n$. Two sets X and Y are diffeomorphic iff $\exists$ a smooth (C^∞, or having continuous partial derivatives of all orders) map with a smooth inverse) map f such that $f : X \to Y$ is bijective and $f^{-1} : Y \to X$ is smooth [18].

bounded. This assumption is made explicitly in [21]. However, it is not necessary, since the parameter d_{LEAVE} ensures that we need not consider local minima outside of the closed disc, or *cell*, with radius $d(S,T)$ centred at T, $B_T(d(S,T))$:

Proposition 2. *All relevant local minima are contained within* $B_T(d(S,T))$.

Proof. By Corollary 4.12 from [21], the distance to T decreases between successive local minima/switching points from Tangent Bug's motion-to-goal mode to its boundary-following mode.[§] Hence we need only show that the first local minimum m_1 lies within $B_T(d(S,T))$, since all subsequent minima will lie closer to the goal. The robot detects m_1 from the point S_1, which terminates the first motion-to-goal segment. Let d_{reach} denote the closest distance to T measured along the blocking obstacle boundary (which contains m_1). Clearly, since the distance to T decreases monotonically along a motion-to-goal segment, we have $d(m_1,T) \leq d_{reach} < d(S_1,T) \leq d(S,T)$. □

Thus, by the **Bolzano-Weierstrass Theorem** [19], the relevant portion of C-space is compact. (To simplify our terminology, we will refer to "the relevant portion of C-space" as $\mathcal{D}$.[¶]) Two important well-known properties of compactness are

1. Every open cover of a compact space K has a finite subcover [19],[29],[68].

2. Every infinite set X in K has at least one *cluster point* in K, that is, there is a point $x \in K$ such that for each open set O containing x, O contains a point in X distinct from x (the **Bolzano-Weierstrass property**) [29],[68].

The relevant portion of freespace (again, to simplify terminology we will refer to this region as $\mathfrak{F}$), a closed subset of $\mathcal{D}$, inherits the properties of being a compact metric space from $\mathcal{D}$. $\mathfrak{F}$ is also a 2-manifold with boundary. However, $\mathfrak{F}$ is not topologically equivalent to a cell, but rather is $\mathcal{D} - \bigcup_i int(O_i)$, where the set $\{O_i\}$ is the set of obstacles intersecting $B_T(d(S,T))$, and for each i, $int(O_i)$ is the largest open set contained in O_i (in other words, if ∂O_i is the boundary of O_i, then

[§]Lemma 6 in Chapter 4 yields an analagous result for the Wedgebug algorithm.
[¶]$\mathcal{D}$ is used in [19] to denote a cell.

$int(O_i) = O_i - \partial O_i$). (See Figure 3.8.) We assume that the set $\{O_i\}$ is finite, i.e., that there are a finite number of obstacles in $\mathcal{D}$. Further, we assume that ∂O_i has finite arclength for every i (i.e., it is *rectifiable*—see Sect. 3.3), and that the boundary is smooth.[‖]

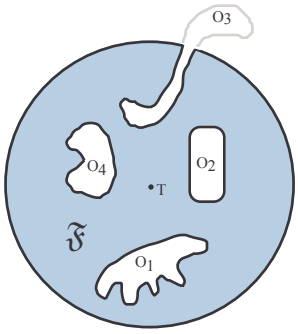


Figure 3.8: A sketch of $\mathfrak{F} = \mathcal{D} - \bigcup_i int(O_i)$.

Consider the function $f_q : \mathfrak{F} \to \mathbb{R}$ such that $f_q(p) = \mathrm{d}(p, q)$, where q is a point in $\mathbb{R}^2$ and $\mathrm{d}(x, y)$ is the Euclidean metric on $\mathbb{R}^2$.[**] Let V be the vector field $-\overrightarrow{\mathrm{grad}}(f_q)$[††]; that is, at every point $p \in \mathfrak{F}$, let

$$V(p) = \Big(-\frac{\partial f_q}{\partial x_1}(p), -\frac{\partial f_q}{\partial x_2}(p) \Big).$$

It can be shown that the zeroes of V are precisely the *critical points* of f_q (i.e., those points where $d(f_q)_p = 0$). Further, x is an *isolated*[‡‡] zero of V iff it is an isolated

[‖] We will later extend our result to continuous boundaries that are not necessarily smooth.

[**] f_T is known as the *distance function with pole q* [55].

[††] The use of the negative gradient echoes the robot's motion to decrease its distance from the goal.

[‡‡] A set of **isolated points** $E \in X$ has the property that for each $c \in E, \exists$ an open set O_c containing c such that $O_c \cap E = \{c\}$ [68].

Note that this is a different use of the term *isolated* than in the statement (which we will denote by $\triangle$) that "any set of isolated points in a compact space is finite" [21]. The definition of this second sense is as follows: A point x is *isolated* iff the set $\{x\}$ is open [29]. The following is a proof of $\triangle$, by Jake Matijevic [44]: Given a compact set K, then for each isolated point $x \in K$, let N_x be the open set $\{x\}$. Then, the set of all N_x can be expanded to be an open cover of K. Since K is compact, every open cover has a finite subcover. Therefore, there must be a finite number of N_x, and thus a finite number of isolated points in K.

However, this second definition does not apply to the situation of critical points (which are isolated iff V maps a neighborhood of each critical point homeomorphically onto a neighborhood of 0 in $\mathbb{R}^2$ [19]; this definition is equivalent to the first definition above). Moreover, the first definition of *isolated* is not sufficient for the statement $\triangle$, as shown by the following counterexample: Consider the closed interval $K = [0, 1] \in \mathbb{R}$. Since this interval is a closed, bounded subset of a Euclidean metric space, it is compact. Consider the sequence $E = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, ...\}$. Clearly, this infinite subset has the limit point 0 (zero), which is in K but not in E. Now we will construct an open set $O_c \in K$ for each $c \in E$ such that $O_c \cap E = \{c\}$. Let O_c be the open interval $(c - \frac{c}{2}, c + \frac{c}{2})$. Thus, we have constructed an infinite set of points in K, all of which are isolated (in the first sense). Clearly, then, the use of the statement $\triangle$ by Kamon, Rimon, and Rivlin in their proof of Lemma 4.1 in [21] is incorrect.

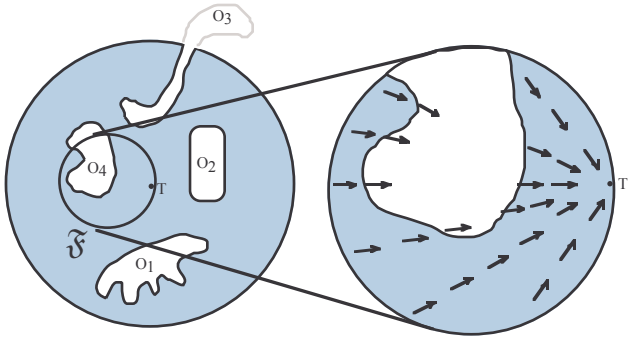


Figure 3.9: A sketch of the vector field V associated with f_T in the $\mathfrak{F}$ from Figure 3.8. Note that V has no zeroes except at T.

critical point of f_q [18].

We can use V to show that if all of the critical points of f_q (in a compact set K where V is continuous on K) are isolated, then there must be a finite number of them.* Suppose not, and let K contain an infinite number of critical points of f_q. Then by compactness, K must contain a cluster point p of the set of critical points. By the continuity of V, we must have $V(p) = 0$; thus, p is a nonisolated critical point of f_q, a contradiction.

Consequentially, we need only show in our case, where $q = T$, that the critical points of f_T are isolated (or that f_T is *non-degenerate* (ND)†, since f_T ND implies that all of its critical points are isolated [55]) on a compact manifold where V is continuous, to prove that there are a finite number of critical points of f_T—and therefore local minima of f_T—on that manifold. Unfortunately, although V is continuous on $\mathfrak{F}$, it does not have any zeroes at all except at the global minimum, T (if $T \in \mathfrak{F}$) (see Figure 3.9). This approach will not find the local minima (or maxima) of $\mathrm{d}(\cdot, T)$, which are points on the manifold boundary where the vector field V is normal to the boundary.

Note: A continuous function $f : X \to Y$ is a *homeomorphism* iff f is bijective on X and f^{-1} is continuous [68]. [18] defines isolated critical points in terms of diffeomorphisms, but we need only that V is continuous for our purposes.

*This proof is originally from [19].

†A critical point a of a function f is *non-degenerate* (ND) iff the Jacobian of f at a is nonvanishing; a function f is ND iff each of its critical points are ND [55].

However, we may consider $\mathfrak{F}$ as a collection of distinct manifolds, where each obstacle boundary (and the boundary of $B_T(\mathrm{d}(S,T))$) is considered as a separate 1-dimensional manifold. (If ∂O_i does not lie entirely within $B_T(\mathrm{d}(S,T))$, then the intersection points $\partial O_i \cap \partial B_T(\mathrm{d}(S,T))$ form separate 0-dimensional manifolds.) This collection of manifolds, along with the open 2-manifold $int(\mathfrak{F})$, is called a *stratification* of $\mathfrak{F}$, and each distinct manifold is called a *stratum*. We redefine a vector field V^* on this stratification as follows: for each $p \in \mathfrak{F}$,

$$V^*(p) = -\overrightarrow{\mathrm{grad}}(f_T|_M) = \Big(-\frac{\partial f_q}{\partial x_1}(p), -\frac{\partial f_q}{\partial x_2}(p) \Big)\Big|_{T_p(M)}$$

where M is the stratum containing p. In other words, the vector field restricted to each 1-dimensional stratum is now the gradient of f_T projected onto the tangent to the obstacle boundary (see Fig. 3.10). It is clear that zeroes of V^* will occur at T (the global minimum of $\mathrm{d}(\cdot,T)$, if $T \in \mathfrak{F}$) and at points on the obstacle boundaries where $-\overrightarrow{\mathrm{grad}}(f_q)$ is normal to ∂O_i. Zeroes of V^* also occur along $\partial B_T(\mathrm{d}(S,T))$ and at the points $\partial O_i \cap \partial B_T(\mathrm{d}(S,T))$, since the latter points have 0-dimensional tangent spaces. We may disregard all zeroes on $\partial B_T(\mathrm{d}(S,T))$, since these (non-isolated zero) points do not represent local minima of $\mathrm{d}(\cdot,T)$. (Most of these points represent local maxima; the obstacle boundary intersection zeroes are artifacts which cannot represent relevant local minima—points m_i which dictate that the planner may switch between motion-to-goal and boundary-following behaviours—because Prop. 2 ensures that $\mathrm{d}(m_i,T) < \mathrm{d}(S,T)$.) We may also disregard T, if $T \notin \partial O_i$ for all i, since it is the goal where the algorithm terminates.

Thus, we restrict our search for critical points (which may correspond to relevant local minima) along the (finite number of) 1-dimensional manifolds ∂O_i. (We include the entire obstacle boundaries, rather than restricting ourselves to the cell $B_T(\mathrm{d}(S,T))$, since if the number of local minima on an obstacle boundary is finite, then the number of local minima on the part of the boundary within $B_T(\mathrm{d}(S,T))$ is also finite.) Since each obstacle boundary is topologically equivalent to S^1 (a closed, bounded subset of a cell), it is compact [19].

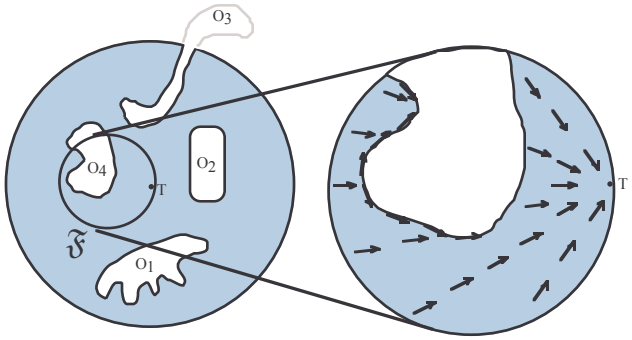


Figure 3.10: A sketch of the vector field V^* associated with f_T and the stratification of $\mathfrak{F}$ given in the text. Note that V^* has zeroes at T and where V is normal to the obstacle boundary.

Since the obstacle boundaries are smooth, we have a situation applicable for the machinery of stratified Morse theory (described in detail in [17]). We will not discuss this theory in detail, but rather note that in this case, it can be shown that f_T is a Morse function on ∂O_i, given a generic environment[‡] [55], [51]. One nice property of Morse functions is that their critical points are non-degenerate[§], and therefore isolated [18], [55]. Since Morse functions are smooth, it follows that V^* is continuous on ∂O_i, and therefore there are a finite number of critical points—and hence local minima—on ∂O_i.

Finally, we have proven:

Lemma 4.1 (from [21])**.** *There are a finite number of isolated local minimum points of* $\mathrm{d}(x, T)$ *over the free configuration space* $(\mathfrak{F})$*.*

This result—that there are a finite number of (relevant) local minima for a generic planar environment containing a finite number of obstacles with smooth (and rectifiable) boundaries—is the same result presented as Lemma 4.1 in [21], with the exceptions that the assumption in [21] that the environment is bounded

[‡]i.e., T is not a *focal point* for any O_i, where a focal point of a manifold M is defined as follows: If the vector $p - q$ is normal to M, and p is a degenerate critical point of the mapping $p \to \|p - q\| : M \to \mathbb{R}$, then q is a focal point of M with base point p [55]. In other words, the environment is considered **not** generic if any part of an obstacle contains an arc whose center of curvature is T.

[§]The definition of a Morse function is a smooth function whose critical points are all non-degenerate [51], [18].

has been replaced here by Prop. 2, the remainder of the assumptions of the proof of Lemma 4.1 in [21] have been clarified, and the statement that any isolated set of points in a compact space is finite, as used by the original proof in [21], has been corrected.

However, we do not wish to restrict ourselves to obstacles with smooth boundaries.

3.4.2 Extension to Piecewise Smooth Boundaries

We relax the assumption that the obstacle boundaries are smooth by allowing the obstacles to have piecewise smooth boundaries, with a finite number of vertices.[¶] We can consider such an obstacle O_i to be made up of several distinct smooth curves C_j, joined at a finite number of vertices x_j (see Fig. 3.11). Treating each open curve $\tilde{C}_j$ as a distinct 1-dimensional manifold (and each vertex x_j as a distinct 0-dimensional manifold) we would like to invoke stratified Morse theory again. Unfortunately, $\tilde{C}_j$ is not compact, so we cannot use our result that a set of isolated critical points on a compact set is finite.

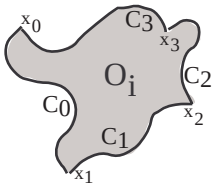


Figure 3.11: An obstacle with piecewise smooth boundaries

Therefore, we will consider each **closed** curve C_j as a distinct entity embedded in $\mathbb{R}^2$. Then, by showing that the number of critical points of $f_T|_{C_j}$ is finite, we will show that the total number of critical points of $f_T|_{\partial O_i}$ is finite. Again, for C_j generic, $f_T|_{C_j}$ is Morse [55], [51]; therefore all critical points of $f_T|_{C_j}$ are isolated. Since C_j is compact, there are a finite number of critical points. Then, since there are a finite number of curves (and vertices, which we count as automatic critical points), there must again be a finite number of local minima. So, we have:

[¶]It is stated in [21], but not proven, that the result in Lemma 4.1 can be extended to obstacles with piecewise smooth boundaries.

$$\begin{aligned}\#\text{ critical pts of } f_T|_{\partial O_i} &\leq \sum_j (\#\text{ critical pts of } f_T|_{C_j}) + \#\text{ vertices of } \partial O_i \\ &= \sum_j (\#\text{ critical pts of } f_T|_{C_j} + 1) \\ &< \infty,\end{aligned}$$

and therefore,

$$\begin{aligned}\#\text{ relevant local minima of } f_T &\leq \sum_i \big(\#\text{ critical points of } f_T|_{\partial O_i}\big) \\ &< \infty.\end{aligned}$$

3.4.3 Extension to Piecewise C^1 Boundaries & Non-Generic Environments

Next, we wish to extend our result to non-generic obstacles, with continuous, piecewise C^1, rectifiable boundaries with a finite number of vertices. Using the same philosophy of decomposition described above, an obstacle boundary ∂O_i consists of a finite number of C^1 curves C_j joined by a finite number of vertices x_j. We consider each closed curve C_j as a separate entity embedded in $\mathbb{R}^2$, defined by $C_j(t) : I \to \mathbb{R}^2$, where $C_j(0) = x_j, C_j(1) = x_{j+1}$. For simplicity in the subsequent discussion, we will denote $f_T|_{C_j}$ simply as f_T, and "critical point" refers to a point in C_j which is a critical point of $f_T|_{C_j}$. We include the endpoints of C_j as critical points in our analysis.

For each nonisolated critical point $y \in C_j$ of f_T, define the (closed) set A_y as the largest path-connected set of critical points containing y. We will denote this set, the "arc A_y," since it composes a circular arc with T at its center of curvature. We will restrict the set of curves we will consider to those which satisfy the constraint:

$$\nexists A_y \text{ such that } A_y = \{y\}.$$

That is, there are no nonisolated critical points which do not belong to a path-connected component containing other critical points. This condition prohibits

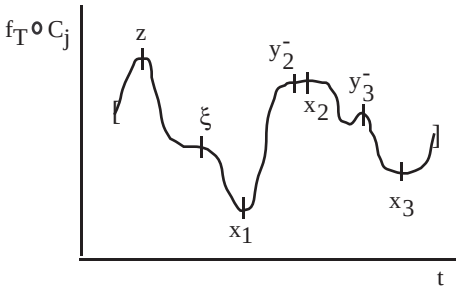


Figure 3.12: Illustration of the definition of y^-. Note that the choice of z for y_1^- in the diagram violates the conditions, since $\exists \xi$ such that $f_T(\xi) > f_T(x)$ and $f_T(\xi - \varepsilon) \leq f_T(\xi)$. Also note that ξ, x_2, and x_3 are contained within arcs.

excessive "wiggliness" in the curve, as exhibited by the "topologists' comb"-like function, $x^3 \sin \frac{1}{x}$, near zero. Let us define a *generalised critical point* (g.c.p.) Z as either an isolated critical point or as an arc A_y (by definition, then, every critical point of f_T is contained within some g.c.p.). Further, we define the "corner points" $\{c\}$ of Z as the set of (at most two) points in Z where either $f_T(c+\varepsilon) < f_T(c)$ and/or $f_T(c-\varepsilon) < f_T(c)$, for any ε small. Let the index of Z, $idxZ$, be the number of "corners" in Z; if Z consists of an isolated critical point c, $idxZ = 2$ if $f_T(c+\varepsilon) < f_T(c)$ and $f_T(c-\varepsilon) < f_T(c)$, for any ε small. If $idxZ = 0$, we say that Z is a local minimum of f_T. If $idxZ = 2$, Z is a local maximum; else, Z is a saddle. We will call $Z \setminus \{c\}$ a *reduced g.c.p.*

Now, for each critical point x (isolated or nonisolated), let I_x be the closed interval containing x and bounded by $y^- = C_j(t^-)$ and $y^+ = C_j(t^+)$, defined as follows: Let t^- be the smallest value such that $f_T(y^-) \geq f_T(x)$ and $f_T(y^- - \varepsilon) \leq f_T(y^-)$ for any ε small, **and** $\nexists \xi \in (y^-, x)$ such that $f_T(\xi) > f_T(x)$ and $f_T(\xi - \varepsilon) \leq f_T(\xi)$ for any ε small (see Fig. 3.12). (Note that if such a ξ exists, it is a critical point of f_T.) Define y^+ similarly: Let t^+ be the largest value such that $f_T(y^+) \geq f_T(x)$ and $f_T(y^+ + \varepsilon) \leq f_T(y^+)$ for any ε small, **and** $\nexists \xi \in (x, y^+)$ such that $f_T(\xi) > f_T(x)$ and $f_T(\xi + \varepsilon) \leq f_T(\xi)$ for any ε small.

Next, define the open interval B_x as follows: Let $I_x = [a, b]$. Then, $B_x = (a, b)$; i.e., B_x is the largest open set contained in I_x. If $x \in Z$ is a local minimum of f_T, then $Z \subseteq B_x$; otherwise, $Z \setminus \{c\} \subseteq B_x$, where $\{c\}$ is the set of (at most two)

"corner points" in Z. With some abuse of language, we call the interval B_x the *basin of attraction* of x for f_T [20]. Intuitively, if we were to let a particle follow the (negative) gradient of $f_T|_{C_j}$, starting at any point in B_x, the particle would end at x (or on an arc A_x containing x such that $f_T(y) = f_T(x)\ \forall y \in A_x$). Note that each B_x contains at most one reduced g.c.p. $Z \diagdown \{c\}$, and since B_x is defined for each critical point x, every $Z \diagdown \{c\} \in B_x$ for some x.

Due to the definition of B_x, the only points ν such that $\nu \notin \bigcup B_x$ are the "corner points." Let $\nu \notin$ any B_x. If $\overrightarrow{\text{grad}}(f_T(\nu)|_{C_j}) = 0$, then $\nu \in Z$ for some generalised critical point Z. Since $\nu \notin \bigcup B_x$, ν must be a corner point. If $\overrightarrow{\text{grad}}(f_T(\nu)|_{C_j}) \neq 0$, then $\nu \in (x^-, x^+)$, where $x^- \in Z^-$, $x^+ \in Z^+$ are the closest critical points on either side of ν (recall that the endpoints of the curve are critical points). By necessity, either $idxZ^- > idxZ^+$, or $idxZ^+ > idxZ^-$. Assume without loss of generality that Z^- is a local maximum. Then, $x^- = C_j(t^-)$ is a "corner point" (and x^+ is not). Since $\nu \in (x^-, x^+)$, and t^- is the smallest value such that $f_T(x^-) \geq f_T(x^+)$ with $f_T(x^- - \varepsilon) \leq f_T(x^-)$ for any ε small, and $\nexists \xi \in (y^-, x)$ such that $f_T(\xi) > f_T(x^+)$ and $f_T(\xi - \varepsilon) \leq f_T(\xi)$ for any ε small (since no critical point intervenes between x^- and x^+), then $\nu \in B_{x^+}$, a contradiction.

The set of B_x for all critical points $x \in C_j$ can be expanded to form an open cover of C_j. For each point $\nu = C_j(t_\nu)$ such that $\nu \notin \bigcup B_x$, it follows from the discussion above that ν is a "corner point" for some g.c.p. Z. Construct an open set $O_\nu = (a, b)$ around ν as follows:

- If $f_T(\nu + \varepsilon) < f_T(\nu)$, for any ε small, then $\exists$ a critical point $x = C_j(t_x)$, $t_x > t_\nu$, such that $f_T(x) < f_T(\nu)$. Let x be the closest such critical point. By the continuity of f_T, $\exists z \in (\nu, x)$ such that $f_T(\nu) > f_T(z) > f_T(x)$ (and note that $z \in B_x$). Let $b = z$.

- If $f_T(\nu + \varepsilon) = f_T(\nu))$, then $\nu \in A_x = [\nu, x]$, for some x. By the Axiom of Choice, we can choose some $z \in A_x$ such that a neighborhood of z lies within A_x. Again, $z \in B_x$, and let $b = z$.

- If $f_T(\nu + \varepsilon) > f_T(\nu))$, for any ε small, then $\exists$ a critical point $x = C_j(t_x)$,

$t_x > t_\nu$, such that $f_T(x) > f_T(\nu)$. Let x be the closest such critical point. By the continuity of f_T, $\exists z \in (\nu, x)$ such that $f_T(\nu) < f_T(z) < f_T(x)$ (and note that $z \in B_\nu$). Let $b = z$. (This case occurs if ν is an isolated saddle point.)

- Define a similarly, using $f_T(\nu - \varepsilon)$.

Therefore, each O_ν contains a single "corner point," and $\{B_x\} \cup \{O_\nu\}$ is an open cover of C_j.

By the compactness of C_j, for every open cover, there exists a finite subcover. Therefore, there must be a finite number of basins of attraction of f_T on C_j, which implies that there are a finite number of reduced g.c.p.'s, $Z \diagdown \{c\}$. Furthermore, there are a finite number of "corner points." Choose one point z_{kj} from the reduced g.c.p. of each of these basins of attraction. Let us now redefine $B_{z_{kj}}$ as $B_{z_{kj}} \cup Z_{z_{kj}}$, where $z_{kj} \in$ g.c.p. $Z_{z_{kj}}$, therefore assigning each "corner point" ν to the unique basin containing $Z \diagdown \{c\}$ where $\nu \in Z$. (This assignment causes B_x to be the true basin of attraction, in the dynamical sense, for the g.c.p. containing x.) Then, there are a finite number of B_x, $\{B_{z_{kj}}\}_1^{N_{ij}}$, (which are clearly disjoint).

Now, the total number of basins of attraction of f_T on ∂O_i, N_i, is bounded by the sum of the number of basins of attraction of f_T on each C_j, N_{ij}, plus the number of vertices (similar to our count of critical points above), which in turn is finite.

The concept of basins of attraction on ∂O_i can be extended to $\mathfrak{F}$. In this case, we again consider the stratification of $\mathfrak{F}$ discussed above, and define $\bar{x}(t)$ as the solution to the equation of motion of a particle acting under the influence of a (non-continuous) force where M is the stratum containing $\bar{x}(t)$ at t. Define $\tilde{B}_x$ as the set of points y such that if $\bar{x}(0) = y$, then as $t \to \infty, \bar{x}(t) \to z \in$ g.c.p. Z_x containing x. Z_x is termed the *ω-limit set* of $\tilde{B}_x$, and points y such that $F(y) = 0$ are called *equilibrium points* [20]. (Note that this definition automatically casts vertices of ∂O_i as equilibrium points; for a vertex x_j, we define its basin of attraction $\tilde{B}_{x_j}$ as the union of the basins of attraction of x_j considered as a point in C_j and in $C_{(j-1) \mod k}$.) Clearly, $B_x \subset \tilde{B}_x$, and $\{\tilde{B}_{z_{kj}}\}$ is disjoint for all $z_{kj} \in C_j$. (Note, $\{\tilde{B}_{z_k}\}_1^{N_i} = \bigcup_1^j \{\tilde{B}_{z_{kj}}\}_1^{N_{ij}}$ covers ∂O_i.) Thus,

$$
\begin{aligned}
\#\text{ relevant basins of } f_T \text{ in } \mathfrak{F} &\leq \sum_i (\#\ \tilde{B}_{z_k} \text{ on } \partial O_i) = \sum_i N_i \\
&\leq \sum_i \big(\sum_j (\#\ B_{z_{kj}} \text{ on } C_j) + \#\text{ vertices of } \partial O_i \big) \\
&= \sum_i \sum_j (N_{ij} + 1) \\
&< \infty,
\end{aligned}
$$

Thus, we have proven:

Lemma 1. *There are a finite number of relevant basins of attraction of* $\mathrm{d}(\cdot, T)$ *over the free configuration space (*$\mathfrak{F}$*), for a finite number of obstacles each having a rectifiable, (finite) piecewise* C^1 *boundary.*

Note that we have not been able to prove in the non-generic case that there are a finite number of local minima, but rather of basins of attraction. Still, this result is enough to show that the Tangent Bug (or Wedgebug) algorithm may switch modes at a finite number of points.

Proposition 3. *Tangent Bug switches from motion-to-goal mode to boundary-following mode when the robot detects that it is in a basin of attraction of* f_T *on some obstacle boundary* ∂O_i*. Further, the algorithm will switch modes at most once for a given basin of attraction.*

Proof. The first statement is merely a restatement of the manner in which Tangent Bug uses range sensors to determine when to switch modes. The second statement is a consequence of the fact that the distance to T decreases between successive (generalised)[‖] local minima of f_T which trigger mode switches along the path, and that the distance to T is constant for all points in the ω-limit set of a given basin. □

Thus, we have completed the proof that Tangent Bug is complete and correct, and have provided a key result for our proof of the completeness and correctness of Wedgebug in Chapter 4.

[‖] i.e., a point z_k chosen from the ω-limit set

Chapter 4

Wedgebug

4.1 Introduction

Useful motion planners for planetary rovers have several key characteristics: they must assume no prior knowledge of the environment, must be sensor-based, robust, complete and correct. They must also operate under severe constraints of power, computational capacity, and the high cost of flight components, which translates into limited memory available on-board the rover. Due to dead reckoning errors, slippage on rough/loose substrate, nonholonomic fine-positioning constraints, and constraints on mission time available, using rover motion to augment sensing is costly. Simultaneously, limited memory, computational capacity, power and time available all argue for minimising the amount of data that must be sensed and processed. Thus, a practical motion planner must utilise the available sensing array in a scheme which efficiently senses only the data needed for motion planning, requires minimal memory to store salient features of the environment, and conserves rover motion.

In the previous chapter, we reviewed TangentBug, an incremental, sensor-based path planner. TangentBug provides the motivation for the work presented here. As discussed in Chapter 3, its world model is streamlined, consisting only of sensed obstacle boundary endpoints. The algorithm is memory-efficient, fairly robust, and conserves robot motion. It is also complete, an essential property of useful planners for autonomous robots. However, some of its assumptions do not apply to

the "rover problem" of navigating in planetary terrain. For example, TangentBug assumes that the robot is modelled as a point, and that obstacles block both motion and sensing. In addition, TangentBug assumes that the robot's sensor provides an omnidirectional view, and does not attempt to minimise the amount of sensed data used for planning.

The current scenario for a rover sensing system consists of a stereo pair of cameras mounted on a pan-able mast. Typically, these cameras have a 30° to 45° field of view (FOV), and the "visible region" associated with these sensors sweeps out roughly a wedge, with limited downrange radius R due to both viewing angle (tilt) and feature resolving ability. (See Fig. 4.1 for an example of data from such a sensing array.) Camera pixels which image features closer to the horizon (hence farther away) have a larger footprint than pixels that image the foreground; simultaneously, obstacles further away are apparently smaller in relative size. These two properties combine to limit the range at which a stereo pair can resolve obstacles

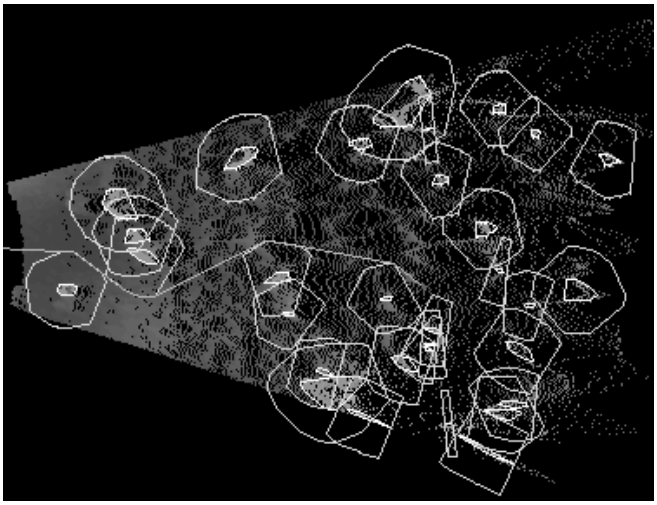


Figure 4.1: Rangemap of a single image from a stereo pair. This image also shows results from the implementation of the "RoverBug" algorithm on Rocky7, tested in the JPL MarsYard (see Chapter 5). Within the wedge are detected obstacles (light grey pixels), delineated by their convex hulls (small white polygons). The "silhouettes" of the convex hulls–essentially the configuration-space obstacles, since the rover is not a point robot–have been generated, represented by the large light grey polygons. A path generated by RoverBug from the rover's current position on the left to a goal near the center of the wedge winds between the forbidden regions.

of a given height, for instance. (Recall that obstacle height is a key parameter for rover navigation.)

Since sensing is limited in both range and angular coverage, we must develop an algorithm which balances the cost of redirecting the sensing array versus the cost of moving the robot, while maintaining completeness and correctness. From the discussion above, it is clear that it is important to not simply pan the sensor array and obtain multiple images that can be synthesised into an omnidirectional view at every step. Rather, the planner should be able to identify the minimal number of sensor scans needed—and which specific areas to scan—to proceed at each step, while avoiding unnecessary rover motion. In the development which follows, it is assumed that the cost of redirecting the sensor array is less than the cost of moving the robot. This assumption is borne out by experience with microrovers in planetary terrain, particularly those whose primary sensors used for higher-level navigation are mounted on a platform capable of being panned independently from the vehicle. Dead reckoning errors, compounded by slippage during motion among rocks and loose substrate, render motion costly. Motion is expensive (compared to sensing) from an energy viewpoint as well, particularly when the sensors used are passive, such as the stereo system used for currently planned Mars rovers. Simultaneously, the limited memory, computational capacity, power, and time available on-board a flight rover for motion planning all argue for minimising the number of views taken.

The resulting planner, a step towards a more practical path planner for planetary microrovers, is the "Wedgebug" algorithm. Wedgebug is complete, correct, and relies solely upon the robot's sensors. Perhaps most importantly, Wedgebug deals with the limited FOV of flight rovers in a manner which is efficient and minimises the need to sense and store data, using autonomous gaze control.

4.2 The Wedgebug Algorithm

A more practical sensing system model for rovers (than TangentBug's omnidirectional view) can be formalised as follows: The rover's sensing array, from position

x, detects ranges within a wedge $W(x, \vec{v})$ of radius R, which sweeps out an angle 2α (> 0) and is centered on the direction $\vec{v}$. (All vectors are assumed to have unit length, unless otherwise specified.) Define C as the arc boundary of $W(x, \vec{v})$ at radius R, and $\partial W(x, \vec{v})$ as the union of the two bounding rays of $W(x, \vec{v})$ (Fig. 4.2). We further define the "interior" of $W(x, \vec{v})$ as $\text{int}(W(x, \vec{v})) = \overline{W(x, \vec{v})} - \partial W(x, \vec{v})$ (N.B., an "interior" point may lie on C). Besides the assumptions already dis-

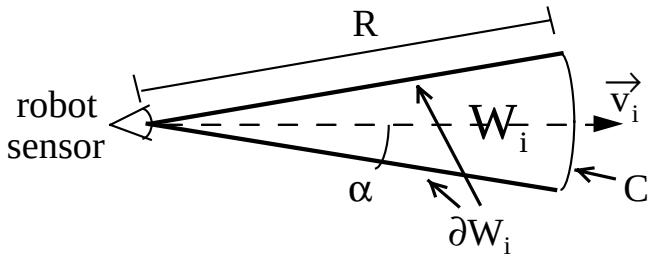


Figure 4.2: Anatomy of a sensor "wedge."

cussed, Wedgebug uses a number of other basic assumptions, as follows: The rover is modelled as a point robot in a 2D environment, and it is assumed to be capable of omnidirectional movement. In addition, the environment is binary–that is, every point in the robot's space is either contained within an impassable forbidden region (an "obstacle") or lies in freespace, $\mathfrak{F}$. The obstacles, in turn, are "wall-like": they block sensing as well as motion (just like walls in an indoor, office-type environment). It can be noted here that these assumptions match those of the TangentBug algorithm, thus isolating the effect of the limited sensor FOV. (Ways to address the non-point volume of the rover, and its more realistic ability to see over many obstacles, are addressed in Chapter 5.) Finally, let $\text{d}(a, b)$ be the Euclidean distance between points a and b.

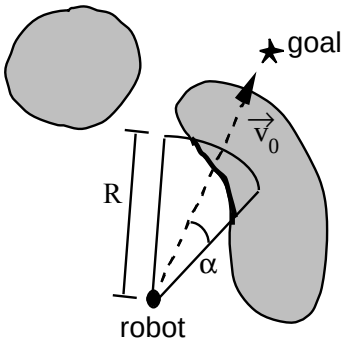


Figure 4.3: The rover's wedge view in relation to its configuration space. The shaded regions are obstacles.

The Wedgebug algorithm operates using two modes, or "behaviours": **motion-to-goal** (abbreviated *MtG*), and **boundary following** (ab-

breviated *BF*), which interact to ensure global convergence to the goal. Each mode is further divided into components which improve efficiency and handle the limited FOV.* The key to the algorithm is the fact that the shortest possible path (assuming only the portions of obstacles within radius R of the robot) passes directly around the *blocking obstacle*, the first obstacle encountered between the robot and the goal (see Chapter 3). Therefore, the Wedgebug algorithm strives to establish the extent of the blocking obstacle (or, rather, enough of the obstacle boundary to choose a preferred direction of traversal around the blocking obstacle) with a minimal number of sensor queries, then move to the appropriate sensed endpoint of the obstacle's boundary, thus conserving robot motion. A high-level sketch of the operation of the Wedgebug algorithm follows: At the beginning of the path sequence, an initialisation step records the parameter $\mathrm{d}_{\mathrm{LEAVE}} = \mathrm{d}(S, T)$, where S is the robot's initial position, and T is the goal. $\mathrm{d}_{\mathrm{LEAVE}}$, set here (and later reset during boundary following), marks the furthest distance away from T that the robot can venture during an *MtG* segment, and is the global parameter which ensures convergence to the goal.

MtG is typically the dominant behaviour. It basically directs the robot to move towards the goal using a local version of the tangent graph, restricted to the visible region (Fig. 4.4). The tangent graph† consists of all line segments in freespace connecting the initial position, the goal, and all obstacle vertices, such that the segments are tangent to any obstacles they encounter [36]. Define the *local tangent graph*, $\mathrm{LTG}(U)$, as follows:

Definition 1. *Let* $\mathrm{LTG}(U)$ *be the* local tangent graph *within the set* U*, defined as the tangent graph restricted to* U*.*

In our case, sensed obstacles appear as continuous countours in the range data, and are differentiated by discontinuities in range measurement (such discontinuities

*It should be noted that due to the tradeoff choices made here, it is possible that a similar robot using TangentBug and omnidirectional sensors could produce a shorter final path. However, Wedgebug will produce the shortest possible (local) path using only the regions sensed, and retains the property of completeness. Section 4.6 discusses several methods to improve optimality of the global path, at the cost of increased sensing.

†also known as the "reduced visibility graph"

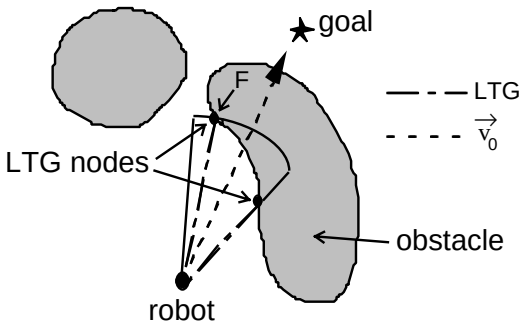


Figure 4.4: The LTG, computed within W_0. The wedge is the same as that shown in Fig. 4.3. The dot/dash lines show the edges of the LTG; the small circles on the obstacle boundary are the two nodes. Note that in this case, since W is directed towards the goal, $G1$ = LTG.

indicate that the ray from the robot in that direction is tangent to an obstacle). An obstacle is further modelled as a thin wall comprising solely its sensed boundary; the endpoints of these "walls" are called the "obstacle vertices." Each endpoint e corresponds to a discontinuity in the range data or to an intersection of a contour with ∂W or C. Note that under the given assumptions (2D environment, with "wall-like" obstacles), LTG(U) consists of all edges between the rover's position x and sensed obstacle boundary endpoints. (LTG(U) also contains edges which are coincidentally tangent between obstacle endpoints, but these edges are never used by Wedgebug, and so are never explicitly constructed.)

MtG works roughly as follows: The robot (at position x) first senses a wedge, $W_0 = W(x, \vec{v}_0)$, where $\vec{v}_0 = \overrightarrow{xT}/\|\overrightarrow{xT}\|$ is the vector from x to the goal (Figs 4.2, 4.3). (All wedges in the subsequent discussion are assumed to subtend a half-angle α.) The planner constructs LTG(W_0) (Fig 4.4). If there are no visible obstacles intersecting the segment of the ray $\overrightarrow{xT}$ lying within the wedge (or the ray is tangent to any obstacles encountered), the planner adds a node T_g to LTG(W_0) at a distance R from x along $\overrightarrow{xT}$, so LTG(W_0) contains a path directly towards T. The planner then searches a subgraph, $G1(W_0) = \{V \in \text{LTG}(W_0) \mid \text{d}(V,T) \leq \min(\text{d}(x,T), \text{d}_{\text{LEAVE}})\}$, for the optimal local subpath. Note that $G1$ essentially contains those nodes in the LTG which lie closer to the goal than the current position. Also, the "optimality" of a path is measured by shortest length–that is, the optimal sub-path will be the shortest local path, restricted to W_0, which brings the robot closest to the goal. Using the criterion discussed in Section 4.3 below, the rover may scan additional wedges as needed, and constructs the LTG in the conglomerate wedge, $\overline{W}(x)$, which

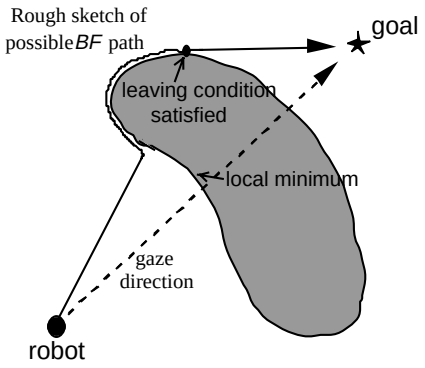


Figure 4.5: Rough sketch of possible *BF* path. The path skirts the boundary of the "blocking" obstacle (the visible obstacle between the robot and the goal) until the leaving condition is satisfied (i.e., when W_0 contains a node of $G1$ which lies closer to the goal than any point so far encountered on the obstacle boundary). (Note: this figure does not illustrate "virtual boundary following," which is discussed in Sect. 4.4.)

is the union of all of the wedges sensed while the robot is at position x.

After executing the resulting subpath, *MtG* begins anew. This behaviour is continued until either the goal is reached, T is deemed unreachable, or the robot encounters a local minimum in $\mathrm{d}(x, T)$, which corresponds to a non-convex blocking obstacle which spans the visible region (Fig. 4.6). In the latter case, the planner switches to *BF*. The objective of this mode is to skirt the boundary of the blocking obstacle (the obstacle whose boundary contains the local minimum), still calculating $\mathrm{LTG}(W_0)$, until one of two events occur: either the robot completes a loop, in which case the goal is unreachable and the algorithm halts; or $\mathrm{LTG}(W_0)$ contains a new subpath toward the goal (the *leaving condition*), which triggers a switch back to *MtG* (Fig 4.5). Of note, no information, other than explicitly recorded points and parameters, are passed between steps.

A (very) high-level depiction of the Wedgebug algorithm in pseudocode follows:

The robot is initially located at S, with goal T.

1. *Initialisation* Global variables:

   Set $\mathrm{d}_{\mathrm{LEAVE}} = \mathrm{d}(S, T)$.

   Set $O_b = \emptyset$, $V_{loop} = \emptyset$, and $\rho^+ =$ "none." (see Sect. 4.3).

   Set **done** = FALSE. /* Goal has not been achieved */

2. Call *MtG*$(S, T, \emptyset)$.

Boolean function *MtG*$(x_{\mathrm{init}}, T, W)$

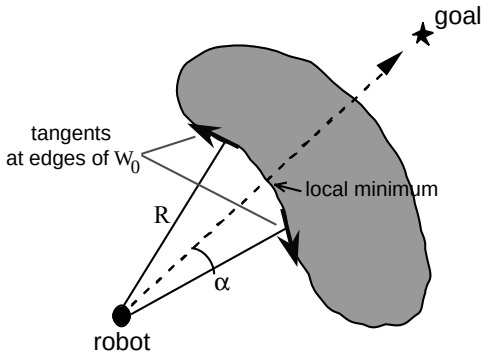


Figure 4.6: Example of inescapable local minimum. Note that the tangents to the obstacle boundary at the edges of $W(x, \vec{v_0})$ slope away from the goal ($\vec{t_F} \cdot \overrightarrow{FT} < 0$).

Boolean function $\mathsf{BF}(x_{\text{init}}, T, \overline{W}, \rho^+, V_{loop}, k)$

Note that the *MtG* and *BF* functions return a Boolean value which indicates whether the goal has been achieved. The *MtG* function is also passed essentially a flag which indicates whether the wedge directed toward the goal from x_{init} has already been sensed (by the prior *BF* function). If the wedge has been sensed, W is set equal to the wedge; otherwise, $W = \emptyset$. Additionally, the *BF* function is passed a third argument, $\overline{W}$, the conglomerate wedge assembled by the immediately preceeding *MtG* function call. By using the already assembled conglomerate wedge, *BF* is spared the necessity of re-sensing an area. Its fourth argument, ρ^+, records the positive direction of traversal around the blocking obstacle, if there is one; otherwise, it is set to "none." The fifth argument, V_{loop}, marks the point on the current blocking obstacle (if there is one) which, if reached, indicates that the robot has executed a loop. (If the robot was not in "sliding" mode prior to the switch to *BF* (see Section 4.3), $V_{loop} = \emptyset$.) Finally, the index k indicates the direction of the last wedge sensed during the prior *MtG* step. The next two sections describe the *MtG* and *BF* modes in more detail, including those aspects which are aimed at efficiency.

4.3 Motion-to-Goal

As described above, the basic idea of *MtG* is to move toward the goal through freespace, $\mathfrak{F}$, using the shortest possible visible path—in other words, the robot assumes that it is only allowed to traverse the area it is currently sensing in its quest to get as close as possible to the goal—in such a manner that its distance to

T is nonincreasing. During this type of behaviour, there are two situations: either the robot can move directly through freespace toward the goal ("direct mode"), or it must skirt the boundary of a *blocking obstacle* while still decreasing its distance from T ("sliding mode"). Further, "sliding mode" contains another submode, "virtual *MtG*," in order to improve efficiency. That is, during normal *MtG*, the planner scans a single wedge view toward the goal to determine whether a path exists. If it is apparent that more information could lead to a shortcut around an obstacle boundary, "virtual *MtG*" scans additional wedges in order to determine the appropriate shortcut, and therefore improve efficiency.

The first actions taken in a new *MtG* segment, after checking to see whether the robot is already at the goal (in which case the algorithm halts and returns TRUE), is to scan a wedge directed toward the goal, $W_0 = W(x, \vec{v}_0)$.[‡] (In the sequel, we use $\overline{W}(x)$ to denote the current composite view, which may consist of a single wedge or multiple wedges.) Next, the planner computes LTG(W_0). If the segment of the line $\overleftrightarrow{xT}$ (connecting the current point with the goal) from x to the intersection with C (the arc boundary of W_0) lies within $\mathfrak{F}$, and $\mathrm{d}(x, T) > R$, add node T_g at this intersection ($\overleftrightarrow{xT} \cap C$) to LTG($W_0$). T_g is the projection of the goal onto the visible wedge. (If $\mathrm{d}(x, T) \leq R$ and no obstacle properly intersects $\overline{xT}$, set $T_g = T$.) Note that T_g does not serve as a goal itself, but rather ensures that a direct path in W_0 exists in the case that there are no visible obstacles between x and T.

Using LTG(W_0), the planner finds the subgraph $G1 = \{V \in \mathrm{LTG}(W_0) \mid \mathrm{d}(V, T) \leq \min(\mathrm{d}(x, T), \mathrm{d}_{\mathrm{LEAVE}})\}$. Nodes in $G1$ lie closer to the goal than the current position, and are closer than the initialisation parameter $\mathrm{d}_{\mathrm{LEAVE}}$. The latter condition ensures that each *MtG* segment is contained within a bounded region, $B_T(\mathrm{d}_{\mathrm{LEAVE}})$, the circle (or "ball") centered at T with radius $\mathrm{d}_{\mathrm{LEAVE}}$. As $\mathrm{d}_{\mathrm{LEAVE}}$ is reset (nonincreasingly) by *BF* segments (Section 4.4), the tightening constraint on succeeding *MtG* segments to remain inside $B_T(\mathrm{d}_{\mathrm{LEAVE}})$ ensures global convergence to the goal (Section 4.5). If $G1 \neq \emptyset$, then next the planner searches $G1 \cup T$ to find the shortest-distance path to the goal. If P is the resultant path, then let P_R be the subpath (contained wholly

[‡]If this wedge has already been scanned by a prior *BF* step, that range data is re-used.

within W_0) comprising the path segments solely between nodes of $G1$. We know (from Chapter 3) that P will pass through either T_g (if $T_g \in G1$) or an endpoint e of a *blocking obstacle* (which intersects the ray $\overrightarrow{xT}$ within W_0); call the point through which the shortest path passes the *focus point*, F (Fig 4.4). The focus point (fixed for each step) serves as the goal for each *MtG* step. Since the subpath for the current *MtG* step is precisely $\overline{xF}$ (see Proposition 4), F simply marks the direction for the robot to travel during this step to minimise its distance to the goal. The position of F within the robot's FOV also determines whether additional wedge views should be taken to improve efficiency.

Proposition 4. *P_R consists solely of the segment $\overline{xF}$.*

Proof. By construction, P is the shortest length path from x to T in $G1 \cup T$. If $T_g \in G1$, clearly the shortest possible path is $\overline{xT}$, which passes through $F = T_g$, QED.

If, on the other hand, $T_g \notin G1$, we know from Proposition 2.1 from [22] (Chapter 3) that the shortest possible path from x to T must skirt the boundary of the *blocking obstacle*, the obstacle sensed in the direction $\vec{v}_0$. Thus, P must pass through a sensed endpoint of the blocking obstacle; call this point F. Assume without loss of generality that there is one other point in $G1$ through which P passes, call it Y. Clearly, we must have $P_R = \overline{xF} + \overline{FY}$.[§]

Since Y is sensed, $\overline{xY} \in \mathfrak{F}$. Invoking the triangle inequality, we must find that either $\overline{xY}$ is shorter than P_R, or F is collinear with x and Y. In the first case, we have a contradiction, since we know that the shortest path must pass through F. So, we must have x, F, and Y collinear. Let *Path1* be the shortest path from F to T along the ("back of the") sensed blocking obstacle boundary (see Figure 4.7), and let *Path2* be $\overline{FY}+$ the

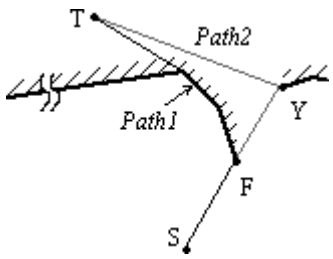


Figure 4.7: Sketch for proof of Proposition 4.

[§]If $P_R = \overline{xY} + \overline{YF}$, then since $\overline{xF} \in \mathfrak{F}$ (because F is sensed), and since by the triangle inequality $\mathrm{d}(x,F) \leq \mathrm{d}(x,Y) + \mathrm{d}(Y,F)$, we have either found a shorter path to the goal—via $\overline{xF}$—in $G1 \cup T$, a contradiction, or Y is collinear with x and F and can be removed from P_R.

remainder of P (i.e., $P - \overline{xY}$). Now, using a similar argument to that used in the proof of Proposition 2.1 (Chapter 3), it can be shown that the polygon bounded by $\overline{FT}$ and *Path1* is included within the polygon bounded by $\overline{FT}$ and *Path2*. Finally, continuing the similar argument, it can be shown that *Path1* is shorter than *Path2*. But this result means that $\overline{xF} + length(Path1) \leq \overline{xF} + length(Path2) = P$, a contradiction. □

Then, simply put, $P_R = \overline{xF}$. The role of F is discussed in more detail below. If $G1 = \emptyset$, the planner switches immediately to *BF*.

If $F = T_g$, the robot simply executes the subpath to F, and starts the next *MtG* step. We call this case a *direct* *MtG* segment, since the robot proceeds through freespace directly towards the goal.

Otherwise, the robot has encountered a blocking obstacle, O, which it must skirt in order to continue towards the goal, in which case *MtG* enters "sliding mode." To ensure that the robot does not backtrack, the planner establishes a traversal direction around the obstacle (i.e., clockwise or counter-clockwise)—call it *positive* (ρ^+)—upon first sensing O. This direction, *positive* (ρ^+), is determined at the end of the *MtG* step in which the blocking obstacle is first sensed, by the final choice of subpath goal: if F is the subpath goal, then $\angle(\overrightarrow{xT}, \overrightarrow{xF}) > 0$ (that is, the direction of rotation from $\vec{v}_0$ to $\overrightarrow{xF}$ is defined as *positive*).[¶] Thereafter, the robot must satisfy the *sliding condition*: it must traverse in the positive direction around O, and may not change direction while following a single (sensed) obstacle's boundary. In subsequent *MtG* steps, after determining the shortest path in $G1 \cup T$, F is chosen as follows: If $F \notin \partial O$, or if F is the sensed endpoint of ∂O in the positive direction (e^+), F is unchanged. Otherwise, F is changed to e^+. At the start of "sliding mode," the planner also records V_{loop}, the sensed endpoint of ∂O in the *negative* direction (e^-). This point is used to detect whether the robot has executed a loop around O during "sliding mode," in which case T is unreachable and the planner halts (and returns FALSE); it is also useful in the case of a switch to *BF* during

[¶]Note that F will not lie on $\overline{xT}$, since then $F = T_g$ and we will not be in "sliding" mode.

"sliding." "Sliding mode" ends when

1. the blocking obstacle is changed (including the case when $F = T_g$),
2. the planner switches to BF, or
3. the robot detects that it has circumnavigated O.[‖]

Below, we delineate the actions of "sliding mode," as determined by the position of F within the visible region:

Case 1: If $F \in \text{int}(W(x, \vec{v}_0))$, there are two cases to consider: either the ray $\overrightarrow{xF}$ is tangent to the blocking obstacle O, or another obstacle is partially obscuring the blocking obstacle's boundary (i.e., $\exists$ a node in $G1$, V, which is collinear[**] with x and F and $\text{d}(x, V) < \text{d}(x, F)$). In the first case, after recording ρ^+ and V_{loop}, the robot simply executes the subpath to F, and starts the next MtG step. This is the case illustrated in Fig. 4.4. (N.B., for purposes of the proof to be given later, the robot never lies directly on an obstacle boundary ∂O, but rather remains a distance ε away.) In the second case, it is possible that the second obstacle is hiding an inescapable local minimum. We will return to this case later.

Case 2: If, on the other hand, $F \in \partial W(x, \vec{v}_0)$; that is, the shortest path around the blocking obstacle, O, intersects the border of the visible region, the planner must inspect the tangent to ∂O at F, $\vec{t}_F$,[††] to see whether the robot will likely be "sliding around" the blocking obstacle while maintaining progress toward the goal, or if it has possibly encountered an inescapable local minimum in $\text{d}(\cdot, T)$. If $\vec{t}_F \cdot \overrightarrow{FT} < 0$, the robot would need to increase its distance from the goal to skirt the obstacle on the subsequent step. So, if allowed, the planner changes F to the opposite sensed endpoint of ∂O, and tests the new F' (see Fig. 4.8). Changing F is not allowed if (1) F has already been changed once at x, or (2) the change would violate the *sliding condition*, the established direction of traversal around an obstacle during

[‖]Case (3) occurs only in the non-generic case when the goal is in the center of a circle or a pathologically positioned/sized regular polygon.

[**]Collinearity is not necessarily subject to noisy measurement in this case, since both obstacle boundary endpoints result from the same discontinuity in range measurement.

[††]For Wedgebug, tangents always point out away from the visible region.

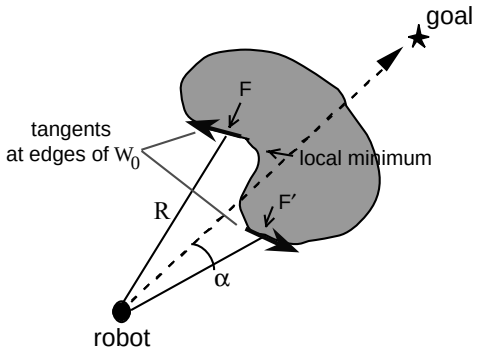


Figure 4.8: Example of escapable local minimum. Note that at F, the tangent slopes away from the goal ($\vec{t}_F \cdot \overrightarrow{FT} < 0$), but not at F′. It would be beneficial to change F in this case to F′.

MtG. If on the second time around, $F' \in \partial W(x, \vec{v}_0)$ and $\vec{t}_{F'} \cdot \overrightarrow{F'T} < 0$ (or F cannot be changed), the robot has encountered an inescapable local minimum in $\mathrm{d}(\cdot, T)$ (Fig. 4.6). Thus, the planner switches to *BF* (described in Sect. 4.4).

In the case that $F \in \partial W(x, \overrightarrow{xT})$, but $\vec{t}_F \cdot \overrightarrow{FT} \geq 0$, the robot must "slide around" the obstacle while progressing toward T. Unfortunately, being close to an obstacle restricts the robot's already-limited view and can result in tiny incremental steps. Thus, in order to efficiently acquire data from the robot's current position and to avoid as much inefficient motion as possible, we add a submode of *MtG*, called "virtual *MtG*." The object of "virtual *MtG*" is to sense (from the current position) additional wedges in the direction the robot will "slide around" the obstacle, and to generate a local shortcut in the robot path. In this manner, the robot uses additional sensing to minimise unnecessary motion.

"Virtual *MtG*" mode directs the sensing array to pan towards F (defining this direction of rotation *positive*, if not already so defined), and to sense the wedge $W_1 = W(x, \vec{v}_1)$, where $\angle(\overrightarrow{xT}, \vec{v}_k) = 2k\alpha,\ k \in \mathbb{Z}$ (that is, W_1 abuts W_0 at F) (Figure 4.9). Let $\overline{W} = W_0 \cup W_1$ (in general, at each position x, $\overline{W}(x) = \bigcup_{sensed} W_i(x)$). The planner computes $G1(\overline{W})$, and finds the new focus point F (using the temporary definition of the *positive* sense of rotation). Let $\partial\overline{W}^+$ be the bounding ray $\vec{r}$ such that $\angle(\overrightarrow{xT}, \vec{r}) > 0$ (i.e., the edge of $\overline{W}$ in the positive direction). If $F \in \partial\overline{W}^+$, "virtual *MtG*" is repeated. This mode ends if one of three conditions is met:

1. $F \in \mathrm{int}(\overline{W})$, in which case the robot has found a suitable shortcut. The robot records ρ^+ and V_{loop} (unless already recorded for this obstacle), executes the

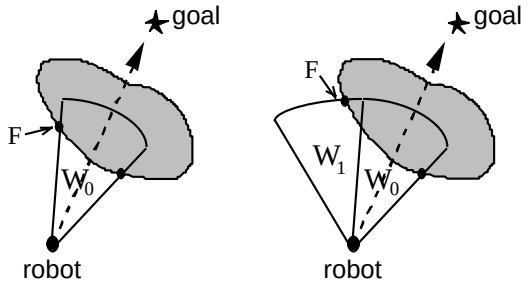


Figure 4.9: Virtual *MtG*. The diagram on the left shows the start of a typical *MtG* step. The small circles depict the nodes of $G1$; the focus point F is indicated with an arrow. Since F lies on the side edge of W_0, "virtual *MtG*" is invoked. The robot senses a second wedge, W_1, which abuts W_0 in the direction of motion around the obstacle (chosen by the selection of F). Since F now lies in the interior of $W_1 \cup W_0$, "virtual *MtG*" stops and the robot will execute the subpath to F.

subpath to F, and begins another *MtG* iteration.

2. $\angle(\overline{xT}, \partial\overline{W}^+) \geq \pi/2$, which means that the rover is sensing part of a region not useful for *MtG*, since G1 contains only nodes closer to T than the robot's current position.

3. $\vec{t}_F \cdot \overrightarrow{FT} < 0$, which indicates that the obstacle boundary is curving back toward x, that is, the robot can no longer "virtually slide" in this direction without losing ground.

In fact, case (2) implies case (3). In these cases, if allowed, the robot changes F as discussed above, and attempts "virtual *MtG*" again. If the second attempt fails, the robot has encountered a local minimum in $\mathrm{d}(\cdot, T)$, and the planner switches to *BF*.

We now return to the case where $F \in \mathrm{int}(W(x, \vec{v}_0))$, but the blocking obstacle boundary is partially obscured by another obstacle. As noted, the obscuring obstacle may be hiding an inescapable local minimum. Thus, as in the case when $F \in \partial W(x, \overrightarrow{xT})$, we check the tangent to ∂O at F. If $\vec{t}_F \cdot \overrightarrow{FT} \geq 0$, "virtual *MtG*" can't help, so the planner records ρ^+ and V_{loop}, executes the subpath, and begins another *MtG* iteration. If $\vec{t}_F \cdot \overrightarrow{FT} < 0$, we change F (if allowed), and check the new F'. If on the second time around, $F' \in \partial W(x, \vec{v}_0)$ or the boundary at F' is similarly

obscured, and $\vec{t}_{F'} \cdot \overrightarrow{F'T} < 0$ (or F cannot be changed), the robot has encountered an inescapable local minimum and the planner switches to *BF*.

Below, we detail the pseudocode version of the the *MtG* behaviour. Recall that we have the global variables $\mathrm{d}_{\text{LEAVE}}, O_b, V_{loop}, \rho^+$, and **done**, explained (and set with their initial values) in Section 4.2. The structure of the *MtG* function is a loop, executed until **done** $\neq$ FALSE. The loop can also be broken in one of two alternate ways: either the algorithm explicitly breaks the loop and halts (in the case that the robot has circumnavigated an obstacle), or it exits the loop to switch to the *BF* behaviour. Note that upon a call to *BF*, the loop *exits*—in particular, once the *BF* function has ended, control does *not* return to the middle of the *MtG* loop. Thus, the structure of the entire program could be thought of as a switch statement (at the start of each step, execute either an *MtG* step or a *BF* step based upon the current behaviour mode), but not as a series of nested loops.

The pseudocode function below utilises several subfunctions, *change_F*, *record_slide*, *slide*, and *reset_slide*. Each of these functions are expanded into their pseudocode representations after the main function, accompanied by a brief description. In addition, *MtG* uses the primitive function $coll(a, b, c)$, which returns a Boolean indicating whether the points $a, b,$ and c are collinear.

Motion-to-Goal: Boolean *MtG*(x, T, W)

While (done = False) {

1m. Set *flag*(F changed) = FALSE. Set $k = 0$. /* k is a counter indexing the current sensed wedge */

2m. *If* $W = \emptyset$, *then* sense $W_0 = W(x, \overrightarrow{xT})$.
Else, set $W_0 = W$; set $W = \emptyset$.

3m. Compute LTG(W_0).
If $\overline{xT} \cap W_0 \in \mathfrak{F}$, AND $\mathrm{d}(x, T) > R$, *then* add $T_g = \overline{xT} \cap C$ to LTG(W_0).
Compute $G1 = \{V \in \text{LTG} \mid \mathrm{d}(V, T) \leq \min(\mathrm{d}(x, T), \mathrm{d}_{\text{LEAVE}})\}$.

4m. *If* $T \in G1$, *then* move to T; set **done = True**; return.

5m. *If* $G1 = \emptyset$,
then call *BF*$(x, T, W_0,$"none," $\emptyset, 0)$. /* Exit *MtG*(); go to (1b). */

6m. *Else*, find the shortest path to goal within $G1 \cup T$, call it P.

Find $P_R = P|_{W_0} = \overline{xF}$. /∗ F is the "focus point" ∗/

Let $\overline{W} = W_0$.

F = *slide*$(x, T, F, \overline{W}, O_b, \rho^+)$. /∗ check the sliding condition ∗/

7m. *If* $F = T_g$, *then* go to (11m).

8m. *If* $V_{loop} \neq \emptyset$ AND $V_{loop} \in \partial O^{sensed\ +}$ (i.e., in the positive direction on the sensed boundary), *then* **break** (return **done = False**).

/∗ loop encountered; halt ∗/

9m. *If* $F \in \text{int}(\overline{W})$, *then*

if $\exists V \in G1$ such that coll(x, V, F) & d$(x, V) <$ d(x, F),

then if O is the *blocking obstacle*, let $\vec{t}_F$ be the tangent to ∂O at F.

If $\vec{t}_F \cdot \overrightarrow{FT} < 0$, *then* $F =$ *change_F*$(x, F, G1, T,$ *flag*$(F$ changed)).

If $F \neq \emptyset$, *then* go to (9m) to check new $\vec{t}_F$.

Else, call $\mathsf{BF}(x, T, \overline{W}, \rho^+, V_{loop}, k)$. /∗ Exit $\mathsf{MtG}()$; go to (1b). ∗/

Else, record_slide$(x, T, F, \overline{W})$; go to (11m).

10m. *If* $F \in \partial W_0$, *then* if O is the obstacle containing F, let $\vec{t}_F$ be the tangent to ∂O at F.

If $\vec{t}_F \cdot \overrightarrow{FT} < 0$, *then* $F =$ *change_F*$(x, F, G1, T,$ *flag*$(F$ changed)).

If $F \neq \emptyset$, *then* go to (9m) to check new $\vec{t}_F$.

Else, call $\mathsf{BF}(x, T, \overline{W}, \rho^+, V_{loop}, k)$. /∗ Exit $\mathsf{MtG}()$; go to (1b). ∗/

Else, begin "Virtual MtG":

If $\rho^+ =$ "none," temporarily define the direction of rotation

from $\vec{v}_0$ to $\overrightarrow{xF}$ as *positive*, P^+; else, set $P^+ = \rho^+$.

Let ∂W^+ be the positive bounding ray of W.

If $\angle(\vec{v}_0, \partial\overline{W}^+) < \pi/2$, *then* $k = k + 1$; sense $W_k = W(x, \vec{v}_k)$,

where $\angle(\overrightarrow{xT}, \vec{v}_k) = 2k\alpha$. Let $\overline{W} = \bigcup^{sensed} W_i(x)$.

Compute LTG$(\overline{W})$, $G1(\overline{W})$, then $F(\overline{W})$.

F = *slide*$(x, T, F, \overline{W}, O_b, P^+)$. Go to (8m).

Else, $F =$ *change_F*$(x, F, G1, T,$ *flag*$(F$ changed)).

If $F \neq \emptyset$, *then* go to (9m) to check new $\vec{t}_F$.

Else, call $\mathsf{BF}(x, T, \overline{W}, \rho^+, V_{loop}, k)$. /* Exit $\mathsf{MtG}()$; go to (1b). */

11m. Execute P_R.

}

End.

The function *change_F* determines whether F can be changed (i.e., F has not already been changed at x, and a change would not violate the *sliding condition*). If allowed, the function returns a new F'; otherwise, it returns $\emptyset$.

nodetype *change_F*$(x, F, G1, T,$ *flag*$(F$ changed$))$ {

1c. *If flag*$(F$ changed$)$, *then* return $F = \emptyset$.

2c. *If* $(\rho^+ \neq$ "none") AND $(\angle(\overrightarrow{xT}, \overrightarrow{xF}) > 0)$, *then* return $F = \emptyset$.

3c. Find $Y =$ the other sensed endpoint of the blocking obstacle O_b.

4c. Set *flag*$(F$ changed$)$ = TRUE; return $F = Y$.

}

Upon entering "sliding mode," the function *record_slide* resets the values of the global parameters O_b, ρ^+ and V_{loop}.

void *record_slide*$(x, T, F, \overline{W})$ {

1r. Let O_b be the *blocking obstacle.*

2r. *If* $\rho^+ =$ "none," let ρ^+ mark the *positive* rotation direction about x,

defined by $\angle(\overrightarrow{xT}, \overrightarrow{xF}) > 0$.

3r. *If* $V_{loop} = \emptyset$, *then:*

Let Visible$(x, \overline{W})$ be the star-shaped set such that $\forall n \in$ Visible$(x, \overline{W})$,

$\overrightarrow{xn} \in \mathfrak{F}$. Let $\partial O_b^{sensed} =$ Visible$(x, \overline{W}) \cap \partial O_b$.

Let e^- be the endpoint of ∂O_b^{sensed} in the *negative* direction.

Set $V_{loop} = e^-$.

}

The function *slide* is the key function in "sliding mode." It enforces the *sliding condition* (if the algorithm is in "sliding mode") by checking the position of F within the robot FOV. Based upon the result, the function either exits "sliding mode," or changes F to satisfy the *sliding condition.*

nodetype *slide*$(x, T, F, \overline{W}, O_b, P^+)$ {

1s. *If*$P^+ =$ "none," *then* return F.

2s. *If* $F = T_g$, *then reset_slide*; return F.

3s. Let Visible$(x, \overline{W})$ be the star-shaped set such that $\forall n \in$ Visible$(x, \overline{W})$,

$\overline{xn} \in \mathfrak{F}$. Let $\partial O_b^{sensed} =$ Visible$(x, \overline{W}) \cap \partial O_b$.

Let e^+ be the endpoint of ∂O_b^{sensed} in the *positive* direction.

4s. *If* $F \notin \partial O_b^{sensed}$, *then reset_slide*; return F.

5s. Return e^+.

}

The function *reset_slide* resets the global parameters associated with "sliding" to their initial ("direct mode") values.

void *reset_slide* {

1e. Set $\rho^+ =$ "none," set $V_{loop} = \emptyset$, and set $O_b = \emptyset$

}

4.4 Boundary Following

The basic idea of *BF*, boundary following, is to skirt the *blocking obstacle* until progress can be made once more toward the goal. As with *MtG*, *BF* is split into two submodes. "Normal *BF*" uses two wedge views, one toward the goal and one in the direction of travel around the obstacle boundary, to determine whether a clear path towards the goal exists while the robot circumnavigates the obstacle. Immediately after a switch from *MtG* to *BF*, however, the robot must determine its direction of travel around ∂O_b, the blocking obstacle. If ρ^+ is not already defined, "Virtual *BF*" is used to take full advantage of the information which can be gleaned at the current distance from the obstacle (arguably more than from a closer range), to choose this direction efficiently (Fig. 4.11). (The primary motivation for "virtual *BF*" is the idea that it is less costly for the robot to swivel its sensors than for the robot to actually move.) In essence, the robot will swing its sensor array back and forth in a prescribed manner, to search for the "best" place to move and begin "normal *BF*."

More precisely, in "virtual *BF*" the robot initially scans the wedge $W_1 = W(x, \vec{v}_1)$, where in this case the positive angular direction is chosen by comparing the tangents to ∂O_b at the intersection with ∂W_0; that is, if $\vec{t}_l, \vec{t}_r$ are the two tangents (at the intersection points e_l (left) and e_r (right), respectively), then if $\vec{t}_l \cdot \vec{v}_0 \geq \vec{t}_r \cdot \vec{v}_0$, then $\angle(\vec{v}_0, \overrightarrow{xe_l}) > 0$. Recall that the vector $\vec{v}_k$, $k \in \mathbb{Z}$, is defined by $\angle(\overrightarrow{xT}, \vec{v}_k) = 2k\alpha$. As before, let $\overline{W} = \bigcup_k^{sensed} W_i(x)$ (in this case, $\overline{W} = W_0 \cup W_1$), and let $\partial\overline{W}^+$ ($\partial\overline{W}^-$) be the bounding ray of $\overline{W}$, $\vec{r}$, such that $\angle(\overrightarrow{xT}, \vec{r}) > 0$ (< 0) (i.e., $\partial\overline{W}^+$ ($\partial\overline{W}^-$) is the edge of $\overline{W}$ in the positive (negative) angular direction). The planner computes LTG($\overline{W}$). (N.B., we use here the LTG, not G1, since during *BF*, the path is allowed to stray farther from the goal than is permitted during *MtG*.) If $\exists$ a node $V \in \text{LTG}(\overline{W}) \cap \partial O_b$ such that $V \in \text{int}(\overline{W})$, the robot moves to V and begins "normal *BF*," first recording two features (Fig. 4.10): d_{reach}, the distance between T and the closest point to T encountered so far on ∂O_b, and $V_{loop} = \partial\overline{W}^- \cap \partial O_b$ (as well as the final direction of traversal around the obstacle, ρ^+). d_{reach} is used to test the leaving condition, and V_{loop} is used to determine whether the robot has executed a loop. If there is no such node V, the planner directs the sensing array to scan $W_{-1} = W(x, \vec{v}_{-1})$, constructs $\overline{W} = W_0 \cup W_1 \cup W_{-1}$, and searches the freshly expanded LTG($\overline{W}$). In this wise, the robot scans back and forth until a suitable node is found, then travels there to begin "normal *BF*." (If the robot has already scanned additional wedges from x when "virtual *BF*" begins, then the robot scans one wedge at a time in a manner which "balances" the visible region with respect to the direction toward the goal, evaluating LTG($\overline{W}$) after each wedge, before reverting to back-and-forth scanning.)

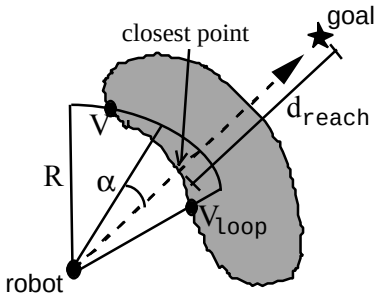


Figure 4.10: Nodes used for *BF* (Boundary Following)

"Virtual *BF*" ends when one of three events are detected:

1. $\exists V \in \text{LTG}(\overline{W}) \cap \partial O_b$ such that $V \in \text{int}(\overline{W})$, and this node is not the result of an occluded boundary (see item (3)). In this case, further scanning will not necessarily yield a shorter path. The robot moves to V, and begins normal

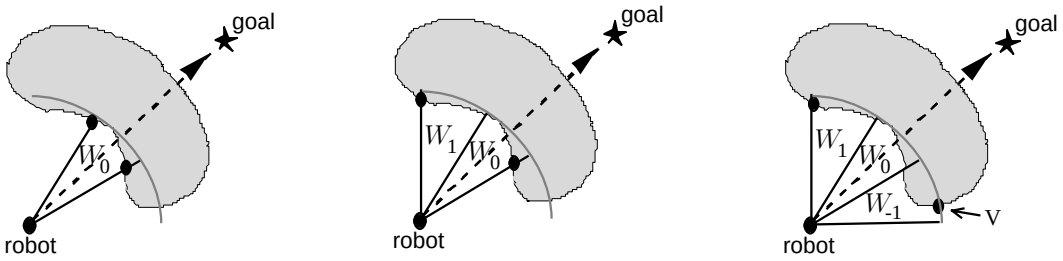


Figure 4.11: "Virtual *BF*." The figure on the left depicts the first part of a "virtual *BF*" step. The nodes of LTG(W_0) are marked with black circles. Since $\nexists V \in \text{int}(\text{LTG}(W_0))$, the robot scans W_1 (center). Again, $\nexists V \in \text{int}(\text{LTG}(W_0 \cup W_1))$, so the robot scans W_{-1} (right). Now, $V \in \text{int}(W_0 \cup W_1 \cup W_{-1})$, so "virtual *BF*" ends.

BF.

2. The next *negative* wedge to be scanned covers a region previously scanned (that is, $k < 0$ and $|\angle(\vec{v}_0, \vec{v}_k)| \geq \pi$). In this case, the robot is trapped by an encircling obstacle, and the algorithm halts.

3. $\exists V \in \text{LTG}(\overline{W})$ with $V \in \text{int}(\overline{W})$, but $V \notin \partial O_{b\ sensed}$. In this case, an obstacle partially obscures the blocking obstacle boundary (see Fig. 4.12). Let $V, V_b \in \text{LTG}(\overline{W})$ be points such that $V, V_b \in \text{int}(\overline{W})$, $V \notin \partial O_{b\ sensed}$, $V_b \in \partial O_{b\ sensed}$, and x, V, V_b are collinear. We call V_b a "framing node," since it "frames" the sensed extent of ∂O_b. If there is only one "framing node" in the current view, the robot scans once more in the opposing direction, and then no matter the outcome, "virtual *BF*" ends. If a node as in item (1) is found, the robot moves there and begins normal *BF*. Otherwise, the robot moves to V_b. If, on the other hand, there are two "framing nodes," V_l and V_r, the rover moves to V_l iff $|\angle(\vec{v}_0, \overrightarrow{xV_l})| > |\angle(\vec{v}_0, \overrightarrow{xV_r})|$). At this point, the rover begins normal *BF*.

In normal *BF*, at the start of each step, the robot senses W_0 (as always, $k = 0$ indicates the direction toward the goal), and searches $G1(W_0)$. *BF* exits here if:

1. $T \in W_0$, in which case the robot moves to T and the algorithm is done, or

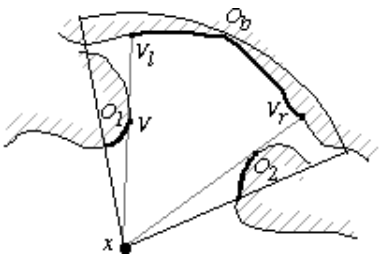


Figure 4.12: Framing nodes. The wedge is directed towards the goal. There are three visible obstacles: O_1, O_2, and O_b, where the latter is the blocking obstacle; and two "framing nodes," V_l and V_r. The nodes of the LTG are marked. Note that $\exists V \in \mathrm{LTG}(\overline{W})$ with $V \in \mathrm{int}(\overline{W})$, but $V \notin \partial O_{b\ sensed}$.

2. $\exists V \in G1(W_0)$ such that $\mathrm{d}(V,T) < \mathrm{d}_{reach}$, the *leaving condition*, in which case the planner resets $\mathrm{d}_{\mathrm{LEAVE}}$ to $\mathrm{d}(V,T)$, and begins a new *MtG* segment.

If neither of these conditions hold, the planner computes $\vec{t}_x$, the tangent to ∂O_b at x (pointing in the established direction, ρ^+), and directs the sensing array to scan $W(x,\vec{t}_x)$. If $V_{loop} \in W(x,\vec{t}_x)$, and $V_{loop} \in$ the connected portion of ∂O_b containing x, the robot has executed a loop—therefore, the goal is unreachable, and the algorithm halts. Otherwise, the planner computes $V \in \partial O_b \cap \mathrm{LTG}(W(x,\vec{t}_x))$ such that $\mathrm{d}(x,V) > \mathrm{d}(x,V')\ \forall V' \in \partial O_b \cap \mathrm{LTG}(W(x,\vec{t}_x))$. The robot records d_{reach}, executes this subpath, then begins a new "normal *BF*" step.

In pseudocode, the *BF* behaviour can be described more explicitly: (For the sake of simplicity in the following, assume that $\overline{W} = W_0$ and $k = 0$.) The main *BF* function uses two subfunctions, *frame* and *record*, which are described afterward.

Boundary Following: Boolean $BF(x, T, \overline{W}, \rho^+, V_{loop}, k)$

1b. *Initialisation* Global variable:

$record(x, T, \rho^+, 0, \overline{W}, O_b, 1)$. /* Set d_{reach} */

"Virtual Boundary Following"

If ρ^+ = "none" { /* Direction of traversal around obstacle is undefined */

2b. Define P^+ such that if $\vec{t}_l, \vec{t}_r$ are the two tangents (at e_l and e_r, respectively), then if $\vec{t}_l \cdot \vec{v}_0 \geq \vec{t}_r \cdot \vec{v}_0$, then $\angle(\vec{v}_0, \overrightarrow{xe_l}) > 0$.

/* Choose a temporary *positive* direction around O (the blocking obstacle), P^+, based upon the tangents to ∂O at $\partial O \cap \partial W_0$ */

3b. Set $k = 1$.

Do { /∗ until the new wedge would overlap an area previously sensed ∗/

4b. Sense $W_k = W(x, \vec{v}_k)$, where $\angle(\overrightarrow{xT}, \vec{v}_k) = 2k\alpha$.

/∗ e.g., W_1 abuts W_0 on the *positive* side chosen in (2b) ∗/

5b. Let $\overline{W} = \bigcup^{sensed} W_i$. Compute LTG($\overline{W}$).

6b. *If* ∃ a node $V \in$ LTG($\overline{W}$) such that $V \in$ int($\overline{W}$), *then:*

if $\exists V' \in$ LTG($\overline{W}$) such that $V' \notin$ int($\overline{W}$) $\cap\, \partial O_b$, *then*

move to *frame*$(x, T, P^+, k, \overline{W}, O_b)$; go to (9b).

/∗ Begin normal *BF* ∗/

Else, record$(x, T, P^+, k, \overline{W}, O_b, 0)$. Move to V; go to (9b).

7b. *Else,* /∗ gaze determination ∗/

if $k < 0$, *then* set $k = |k| + 1$.

Else, set $k = -k$.

} until ($k < 0$ and $|\angle(\vec{v}_0, \vec{v}_k)| \geq \pi$) /∗ the new wedge would overlap an area previously sensed ∗/

8b. Return **done = False**. /∗ Robot is encircled by an obstacle ∗/

}

"Normal" Boundary Following

(ρ^+ is defined) /∗ Direction around obstacle established. Note that $x \in \partial O$ (i.e., x is within ε of ∂O), since ρ^+ not recorded by *MtG* unless robot moves to obstacle boundary ∗/

While ! done {

9b. *If* $W_0 \notin \overline{W}$, *then* sense $W_0 = W(x, \vec{v}_0)$. Let $\overline{W} = \bigcup^{sensed} W_i$.

10b. Compute $G1(\overline{W})$ (as in (2m)).

11b. *If* $T \in G1(\overline{W})$, *then* move to T; return **done = True**.

12b. *Else, if* $\exists V \in G1(\overline{W})$ such that $\mathrm{d}(V, T) < \mathrm{d}_{reach}$ (the *leaving condition*), *then* set $\mathrm{d}_{\mathrm{LEAVE}} = \mathrm{d}(V, T)$. Call *reset_slide* (see *MtG* pseudocode).

Call *MtG*(x, T, W_0). /∗ Exit *BF*; goto (1m). ∗/

13b. Let $\vec{t}_x$ be the tangent to ∂O at x^+. /∗ i.e. pointing in the *positive* direction according to ρ^+ ∗/

14b. *If* $W_{\vec{t}_x} \notin \overline{W}$, *then* sense $W_{\vec{t}_x} = W(x, \vec{t}_x)$. Let $\overline{W} = W_{\vec{t}_x} \cup \left(\bigcup_{i>0} W_i\right)$.

15b. *If* $V_{loop} \in \overline{W} \cap \partial O_b$, *then* return **done = False**.

/∗ the robot completed a loop, seeing the entire obstacle boundary; goal is **unreachable** ∗/

16b. Compute LTG($\overline{W}$). Find $\{V \in \partial O_b \cap \mathrm{LTG}(\overline{W}) \mid \mathrm{d}(x, V) > \mathrm{d}(x, V'),$ $\forall V' \in \partial O_b \cap \mathrm{LTG}(\overline{W})\}$. /∗ i.e., the farthest the robot can advance along ∂O_b in this view. ∗/

17b. *record*$(x, T, \rho^+, 0, \overline{W}, O_b, 0)$.

18b. Move to V.

}

End.

The function *frame* is used to determine the final goal of "virtual *BF*" when the robot has detected a "framing node." In both this function and in *record*, let Visible$(x, \overline{W})$ be the star-shaped set such that $\forall n \in \mathrm{Visible}(x, \overline{W}), \overrightarrow{xn} \in \mathfrak{F}$. Again, *coll*$(a, b, c)$ checks for collinearity of a, b, c. V^+_{frame} and V^-_{frame} are the nodes in the *positive* and *negative* direction, respectively, which obscure the blocking obstacle (i.e., $V^+_{frame} \notin \partial O_b$). Having encountered one such node, this function scans a single wedge in the opposing direction, then chooses an appropriate subgoal for "virtual *BF*." If there are already two such nodes, the function does not scan any additional wedges, but rather chooses a subgoal using the available information.

nodetype *frame*$(x, T, P^+, k, \overline{W}, O_b)$ {

1f. Let *flag* $= 0$. Set $V^+_{frame} = \emptyset$. Set $V^-_{frame} = \emptyset$.

2f. Let $\partial O_b^{sensed} = \mathrm{Visible}(x, \overline{W}) \cap \partial O_b$.

Let ∂O_b^+ (∂O_b^-) be the sensed boundary endpoint in the *positive* (*negative*) direction.

3f. *If* $\partial O_b^+ \in \mathrm{int}(\overline{W})$, *then*

if $\exists V \in \mathrm{LTG}(\overline{W}) - \partial O_b^+$ such that $V \in \mathrm{int}(\overline{W})$ and $\mathrm{coll}(x, V, \partial O_b^+)$, *then* let $V^+_{frame} = \partial O_b^+$.

Else, record$(x, T, P^+, +1, \overline{W}, O_b, 0)$. Return ∂O_b^+.

4f. *If* $\partial O_b^- \in \text{int}(\overline{W})$, *then*

if $\exists V \in \text{LTG}(\overline{W}) - \partial O_b^-$ such that $V \in \text{int}(\overline{W})$ and $\text{coll}(x, V, \partial O_b^+)$,

then let $V_{frame}^- = \partial O_b^-$.

Else, record$(x, T, P^+, -1, \overline{W}, O_b, 0)$. Return ∂O_b^-.

5f. *If* $V_{frame}^+ \neq \emptyset$, *then*

if $V_{frame}^- \neq \emptyset$, *then*

if $|\angle(\vec{v}_0, \overrightarrow{xV_{frame}^+})| \geq |\angle(\vec{v}_0, \overrightarrow{xV_{frame}^-})|$,

then record$(x, T, P^+, +1, \overline{W}, O_b, 0)$. Return V_{frame}^+.

Else, record$(x, T, P^+, -1, \overline{W}, O_b, 0)$. Return V_{frame}^-.

else, flag $= 1$. Go to (6f).

else, flag $= 1$. Go to (7f).

6f. *If flag* $\neq 1$ AND $|\angle(\vec{v}_0, \vec{v}_k)| < \pi$, *then* /* Recall $\angle(\overrightarrow{xT}, \vec{v}_k) = 2k\alpha$. */

if $k > 0$, *then* sense $W_{new} = W(x, \vec{v}_{-k})$. Go to (8f).

else, sense $W_{new} = W(x, \vec{v}_{k-1})$. Go to (8f).

Else, record$(x, T, P^+, +1, \overline{W}, O_b, 0)$. Return V_{frame}^+.

7f. *If flag* $\neq 1$, *then*

if $k < 0$, *then* sense $W_{new} = W(x, \vec{v}_{|k|+1})$.

else, sense $W_{new} = W(x, \vec{v}_{k+1})$.

Else, record$(x, T, P^+, -1, \overline{W}, O_b, 0)$. Return V_{frame}^-.

8f. Compute $\text{LTG}(W_{new} \cup \overline{W})$. Go to (2f).

}

The function *record* sets the values of the parameters associated with *BF*. Note that O_b marks the "state" of recognising that the robot is following a single obstacle boundary (i.e., a continuous curve in the range data).

void *record*$(x, T, P^+, k, \overline{W}, O_b,$ *flag*$)$ { /* Records $\text{d}_{reach}, V_{loop}$, and ρ^+ */

1r. Let $\partial O_b^{sensed} = \text{Visible}(x, \overline{W}) \cap \partial O_b$.

2r. Let $N = \{n \in \partial O_b^{sensed} \mid \text{d}(n, T) < \text{d}(m, T), \forall m \in \partial O_b^{sensed}\}$.

Set $\text{d}_{reach} = \min(\text{d}_{reach}, \text{d}(N, T))$.

If flag ≠ 1 {

3r. *If* $V_{loop} = \emptyset$, *then:*

Let $\partial\overline{W}^{+}$ ($\partial\overline{W}^{-}$) be the bounding ray of $\overline{W}$ in the *positive* (*negative*) direction (according to P^{+}).

if $k > 0$, *then* set $V_{loop} = \partial\overline{W}^{-} \cap \partial O_b^{sensed}$.

Else, set $V_{loop} = \partial\overline{W}^{+} \cap \partial O_b^{sensed}$.

4r. *If* ρ^{+} = "none," *then*

define ρ^{+} such that $\angle(\vec{v}_0, \vec{v}_k) > 0$.

}

}

The Wedgebug algorithm thus deals with the limited FOV of the robot in an efficient manner. The "virtual" submodes both take advantage of the lower cost of panning the sensor array over actual motion, while minimising the number of views required at each step.

4.5 Proof of Completeness

The proof of completeness of the Wedgebug algorithm follows. It is analagous to the TangentBug completeness proof, with a few differences in technical details due to the limited robot FOV. The basic concept is that a path generated by the Wedgebug algorithm consists of a finite number of motion segments, where a new segment is begun when the algorithm switches between its motion-to-goal and boundary following behaviours. Each segment can be shown to comprise a finite number of subsegments, and each of these subsegments is finite in length. Thus, the entire path has finite length.

We first give an intuitive sketch of the proof. The Wedgebug algorithm switches from *MtG* to *BF* iff the robot senses an inescapable local minimum in $\mathrm{d}(x, T)$ (that is, an obstacle completely blocks the robot's FOV). Clearly, the number of *MtG* $\rightarrow$ *BF* switches is limited by the total number of local minima of $\mathrm{d}(\cdot, T)$ in the robot environment. In the case where the local minima are not isolated, we can show that the number of *MtG* to *BF* switches is limited by the total number of basins

of attraction of $\mathrm{d}(\cdot, T)$. Furthermore, since nodes in $G1$ (used to plan the *MtG* subpaths) are constrained to lie within $\mathrm{B}_T(\mathrm{d}_{\mathrm{LEAVE}})$, and since *BF* can switch to *MtG* only at points L_i such that $\mathrm{d}(\mathrm{L}_i, T) < \mathrm{d}_{\mathrm{LEAVE}} < \mathrm{d}(S, T)$, any points where *MtG* $\rightarrow$ *BF* behaviour switching points are restricted to (the closed ball) $\mathrm{B}_T(\mathrm{d}(S, T))$. It was shown in Chapter 3 that there are finitely many disjoint basins of attraction for the distance function in the relevant portion of the robot's environment. Hence, there are a finite number of possible *MtG* $\rightarrow$ *BF* switches, so in particular there is a last *MtG* segment (if no *BF* segment terminates the path either at $T \in \partial O$ for some O or by completing a loop around an obstacle O). Thus, Wedgebug terminates after finite path length.

We now consider the details of Wedgebug behaviour. To define the endpoints of each type of motion segment, we use the terminology listed here (which is an extension of that used by Kamon, Rimon, and Rivlin [21]):

- H_i point where robot first senses obstacle i
- D_i point where robot leaves obstacle i (i.e., the *blocking obstacle* changes)
- S_i switch point between *MtG* & *BF* segments (*MtG* $\rightarrow$ *BF*) (associated with obstacle i)
- M_i "local minimum" point associated with S_i (i.e., the point $\overrightarrow{\mathrm{S}_i T} \cap \partial O_i$)
- L_i point where *BF* leaving condition is met on obstacle i (switch point between *BF* $\rightarrow$ *MtG*)
- P_i point where loop is detected on obstacle i ($\mathrm{d}(\mathrm{P}_i, V_{loop}) \leq R$)
- F focus point (point toward which robot is currently heading)

Note that for *MtG*, F is either an endpoint of a segment of sensed obstacle boundary, or $\mathrm{F} = T_g$.

We delineate every possible type of segment which the Wedgebug algorithm can generate. Next, we show that each type of segment has finite length.

Types of *MtG* segments:

direct: $[\mathrm{L}_i, \mathrm{H}_{i+1}]$ $[\mathrm{D}_i, \mathrm{H}_{i+1}]$ Result when the robot leaves obstacle i, either from a sliding *MtG* segment or from a *BF* segment,

and senses (blocking) obstacle $i+1$.

$[\mathrm{L}_i, \mathrm{S}_{i+1}]$ Result when the robot leaves obstacle i, either

$[\mathrm{D}_i, \mathrm{S}_{i+1}]$ from a sliding *MtG* segment or from a *BF* segment, and switches to *BF* at obstacle $i+1$.

Note: In direct segments, motion is directed towards the goal ($\mathrm{F} = T_g$).

sliding: $[\mathrm{H}_i, \mathrm{D}_i]$ Result when the focus point F slides around

$[\mathrm{H}_i, \mathrm{P}_i]$ obstacle i, until either robot leaves the

$[\mathrm{H}_i, \mathrm{S}_i]$ obstacle, encounters a loop, or switches to *BF*.

Note: In sliding segments, motion is directed toward an endpoint of the obstacle boundary. This focus point, F, "slides" along the obstacle boundary until either the algorithm switches to boundary following, or the blocking obstacle is changed.

Types of *BF* segments:

$[\mathrm{S}_i, \mathrm{L}_i]$ Result when the robot either satisfies the leaving condition,

$[\mathrm{S}_i, \mathrm{P}_i]$ or encounters a loop.

Note: "Virtual *BF*," if used during a given *BF* segment, comprises only the initial step of the segment. It serves to define the direction of traversal around the obstacle; once that direction is defined for a given *BF* segment, "virtual *BF*" is never again called during that segment. In addition, between "virtual *BF*" and sensing along the obstacle tangent, no part of the obstacle boundary between P_i and L_i is missed.

A roadmap of the proof is as follows: We first show that each type of segment, *BF* and *MtG*, has finite length. In these proofs, we use the notation $meas(A, B)$ to denote the (arc)length of the path between A and B, considered as a rectifiable curve in $\mathbb{R}^2$. In many cases, this involves the arclength of a portion of an obstacle boundary. We denote the perimeter of an obstacle O by meas(∂O). The proof for *BF*

segments is straightforward, but we must break the *MtG* segments into two types of subsegments, "direct" and "sliding." The proof for "sliding" *MtG* subsegments requires more work than for "direct," as we must derive the path length from the length of the path of the focus point F. Then, we put together the types of *MtG* subsegments to show that an entire *MtG* segment has finite length.

Next, we show that the distance of the robot from the goal is nonincreasing during *MtG* segments, and that the distance to T is strictly decreasing between successive (generalised)[‡‡] local minima which trigger mode changes along the path. Then we show that Wedgebug switches modes when the robot is in a basin of attraction of a generalised local minimum of the distance function, and that the algorithm will switch only once for a given basin; hence the number of possible mode switches is limited by the number of basins. Therefore, we can show that Wedgebug terminates after traversing a finite length path, and can give a (conservative) upper bound on the path length.

Lemma 2. *BF segments are finite length.*

Proof. There are two cases: a) Let us first consider the path segment $[\mathrm{S}_i, \mathrm{L}_i]$ (see Fig. 4.13). In the usual case, this segment can be considered a shortcut, compared to the path which would be taken by a robot with contact sensors executing the TangentBug algorithm. If we imagine that the robot's sensor range $R \to 0$, the resulting TangentBug path, called the "contact sensor equivalent" path, consists of two pieces: $[\mathrm{S}_i, \mathrm{M}_i]$, and $[\mathrm{M}_i, \mathrm{L}_i]$, where M_i is the local minimum (along the line $\overleftrightarrow{\mathrm{S}_i T}$) associated with S_i. Let N designate the point where the robot actually touches ∂O_i during this *BF* segment. By the triangle inequality,

$$\begin{aligned} \mathrm{meas}([\mathrm{S}_i, \mathrm{L}_i]) &= \mathrm{d}(\mathrm{S}_i, N) + \mathrm{meas}([N, \mathrm{L}_i]) \\ &\leq \mathrm{d}(\mathrm{S}_i, \mathrm{M}_i) + \mathrm{meas}([\mathrm{M}_i, N]) + \mathrm{meas}([N, \mathrm{L}_i]). \end{aligned}$$

[‡‡]See Chapter 3, Sect. 3.4

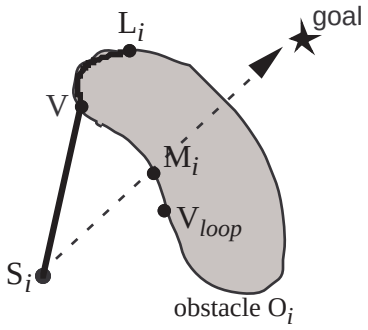


Figure 4.13: Points of interest for proof that boundary following segments are finite length.

We have that (using the appropriate value for $\mathrm{d_{LEAVE}}$)

$$\begin{aligned} \mathrm{d}(\mathrm{S}_i, \mathrm{M}_i) &\leq \mathrm{d}(\mathrm{S}_i, T) \leq \mathrm{d_{LEAVE}} \\ &\leq \mathrm{d}(S, T) < \infty \text{ (by assumption).} \end{aligned}$$

Also, since M_i, N, and L_i all lie on ∂Oi, and the segments $\overline{\mathrm{M}_i N}$ and $\overline{N\mathrm{L}_i}$ do not overlap; and since the obstacle perimeter, $\mathrm{meas}(\partial Oi)$, is finite, we have

$$\mathrm{meas}([\mathrm{M}_i, N]) + \mathrm{meas}([N, \mathrm{L}_i]) = \mathrm{meas}([\mathrm{M}_i, \mathrm{L}_i]) \leq \mathrm{meas}(\partial Oi) < \infty.$$

Thus, this segment is (crudely) bounded by

$$\begin{aligned} \mathrm{meas}([\mathrm{S}_i, \mathrm{L}_i]) &\leq \mathrm{d}(\mathrm{S}_i, \mathrm{M}_i) + \mathrm{meas}([\mathrm{M}_i, N]) + \mathrm{meas}([N, \mathrm{L}_i]) \\ &\leq \mathrm{d}(\mathrm{S}_i, \mathrm{M}_i) + \mathrm{meas}([\mathrm{M}_i, \mathrm{L}_i]) \\ &\leq \mathrm{d}(\mathrm{S}_i, T) + \mathrm{meas}(\partial Oi) \\ &\leq \mathrm{d_{LEAVE}} + \mathrm{meas}(\partial Oi) < \infty. \end{aligned}$$

Of course, we also have $\mathrm{d}(\mathrm{S}_i, \mathrm{M}_i) \leq \mathrm{d}(\mathrm{S}_i, N) \leq R$, where R is the sensing range, and $R < \infty$ by definition. We will subsequently use this fact to determine a bound on the entire pathlength.)

b) Let us now consider $[\mathrm{S}_i, \mathrm{P}_i]$. Similarly, this path can be considered a shortcut relative to its "contact sensor equivalent path," which consists of the two pieces, $[\mathrm{S}_i, \mathrm{M}_i]$, and $[\mathrm{M}_i, \mathrm{M}_i]$ (measured around the obstacle boundary). Thus, $\mathrm{meas}([\mathrm{S}_i, \mathrm{P}_i]) \leq \mathrm{d}(\mathrm{S}_i, \mathrm{M}_i) + \mathrm{meas}(\partial Oi) < \infty$. □

Note that it is possible to have $S_{i+1} = L_i$ for some i. That is, the leaving condition could be satisfied, and then the robot immediately switches back to *BF*. Even if the robot happens to end up following the same obstacle boundary, once the leaving condition is satisfied, the first *BF* segment ends. (In effect, the two *BF* segments are separated by an *MtG* segment with path length = 0. Note that even with an *MtG* segment with zero path length, there is at most one *BF* $\rightarrow$ *MtG* switch per *BF* segment, and Corollary 2 shows that there is at most one *MtG* $\rightarrow$ *BF* switch per basin of attraction, thus (with Lemma 1) prohibiting infinite switching between modes.)

The analysis of *MtG* segments is not as straightforward as the analysis of *BF* segments. The difficulty lies in the fact that the robot has a limited sensor range, and thus may sense new obstacles as the robot progresses towards the goal. While the robot does not sense any (blocking) obstacles, its motion is directed toward the goal (F, the focus point of motion, equals T_g). However, once the robot senses one or more obstacles, the focus point F may change to be one of the endpoints of the sensed obstacle boundaries. If this occurs, the robot enters "sliding mode," which persists until 1) the blocking obstacle changes (or no longer blocks direct progress toward the goal), 2) the algorithm switches to *BF*, or 3) the robot encounters a loop.

Each *MtG* segment can be resolved into a series of subsegments differentiated by the position of F. Where F = T_g, we say that the subsegment is "direct" (i.e., motion is directly towards the goal); otherwise, F lies on an obstacle boundary, and we say the subsegment is "sliding" (since the focus point F "slides" along the sensed obstacle boundary). The analysis of "direct" subsegments follows:

Lemma 3. *"Direct" MtG subsegments are finite length.*

Proof. Since motion along these segments is directed straight towards the goal, we have trivially that for "direct" subsegment $\overline{x_i y_i}$ (i.e., the subsegment of the path between points $x_i, y_i \in \mathfrak{F}$),

$$\text{meas}([x_i, y_i]) = \text{d}(x_i, y_i) \leq \text{d}(x_i, T) \leq \text{d}(S, T) < \infty \text{ (by assumption).}$$

Further, "direct" segments can be broken into two pieces, call them $\overline{a_i b_i}$ and $\overline{b_i y_i}$. If the segment does not immediately follow a *BF* segment, then $a_i = b_i = x_i$. Otherwise, let $a_i = \mathrm{L}_i$ and $b_i = F(\mathrm{L}_i)$ be the initial and final points of the first step of the segment—that is, the first sensing/motion cycle of this "direct" subsegment—respectively. We show that each $\overline{b_i y_i}$ subsegment begins at least as close to the goal as the prior *MtG* subsegment ended.

If $\overline{x_{i-1} y_{i-1}}$ is the preceeding *MtG* subsegment, we have two cases. If the previous subsegment immediately adjoins the current subsegment, $b_i = y_{i-1}$. However, if a *BF* segment intervenes, the parameter d_{reach} ensures that x_i lies within the distance R of a point V which is closer to the goal than any point the robot reached while following the obstacle boundary, and $\mathrm{d}_{\mathrm{LEAVE}}$ ensures that $\mathrm{d}(F(x_i), T) \leq \mathrm{d}(V, T)$. If we call the closest point reached during the *BF* segment x_{BF}, we know that

$$\mathrm{d}(x_{BF}, T) \leq \mathrm{d}(\mathrm{M}_i, T) < \mathrm{d}(\mathrm{S}_i, T),$$

since M_i is closer to the goal than S_i, and M_i is contained in the set of points "reached" by the robot on the obstacle boundary. Finally, $y_{i-1} = \mathrm{S}_i$: that is, the final point of the previous *MtG* subsegment is precisely the switching point marking the start of the intervening *BF* segment. Therefore, since $b_i = F(x_i)$, we have

$$\mathrm{d}(b_i, T) < \mathrm{d}(y_{i-1}, T).$$

Since each $\overline{by_i}$ subsegment begins closer to the goal than the prior *MtG* subsegment ended, the sum of the path lengths of all the $\overline{by_i}$ segments is $\leq \mathrm{d}(S, T)$. Imagine rotating the set of all $\overline{by_i}$ subsegments (over the complete path) as if they were on a nested set of dials, so all of these subsegments line up with one another (see Fig. 4.14). There will be no overlap, only possible dropouts (holes in the line) where sliding or *BF* segments occur. In addition, each $\overline{ab}$ subsegment has path length $\leq R$, and has nonzero length only immediately following a *BF* segment. Since each *BF* segment is uniquely identified with a local minimum, clearly, then, we have the

bound

$$\sum_{i}^{\text{“direct” subsegments}} \text{meas}([x_i, y_i]) = \sum_i (\text{d}(a_i, b_i) + \text{meas}([b_i, y_i]))$$

$$= \sum_i \text{d}(a_i, b_i) + \sum_i (\text{meas}([b_i, y_i])) \leq R \times (\#Minima) + \text{d}(S, T).$$

□

The “sliding” segments take a bit more work. Each sliding segment, combining “virtual *MtG*” and “sliding *MtG*” steps, is defined by a single blocking obstacle, and a fixed sliding direction. The following corollary, originally presented in [21]*, is used in Lemma 4 to show that the path length of a single sliding segment is finite.

Corollary 1. *During MtG, the blocking obstacle may be changed only at a finite number of tangent points on each obstacle.*

Proof. The blocking obstacle will only be changed when the robot has a clear line of sight toward the target, relative to the current blocking obstacle: i.e., when the line $\overleftrightarrow{xT}$ is tangent to the current blocking obstacle. The number of points on each obstacle O tangent to rays radiating from the target is finite. Although it is not proven in [21] that the number of these “tangent points” x_t is finite, note that the x_t are local extrema on ∂O of the “radial distance function” g_T. Define g_T as follows: if $x = (r, \theta)$ are the goal-centric polar coordinates of a point x, then $g_T(x) = \theta$. Then, g_T is a scalar function, as is the distance to the goal. Therefore, by applying the definition of generalised critical points, and by the same argument as in Chapter 3, Section 3.4.3, it can be shown that there are a finite number of local generalised critical points of g_T on ∂O. Further, note that the blocking obstacle is changed at most once per generalised critical point: that is, we say that the robot is still following the obstacle boundary, even if it is moving directly towards the goal, if the robot remains within distance ε of the boundary. Thus, the blocking obstacle is

*The original statement of this corollary appears in [21] as Corollary 4.8; however, the proof presented there is incomplete. The proof we present here is applicable to piecewise C^1 obstacles.

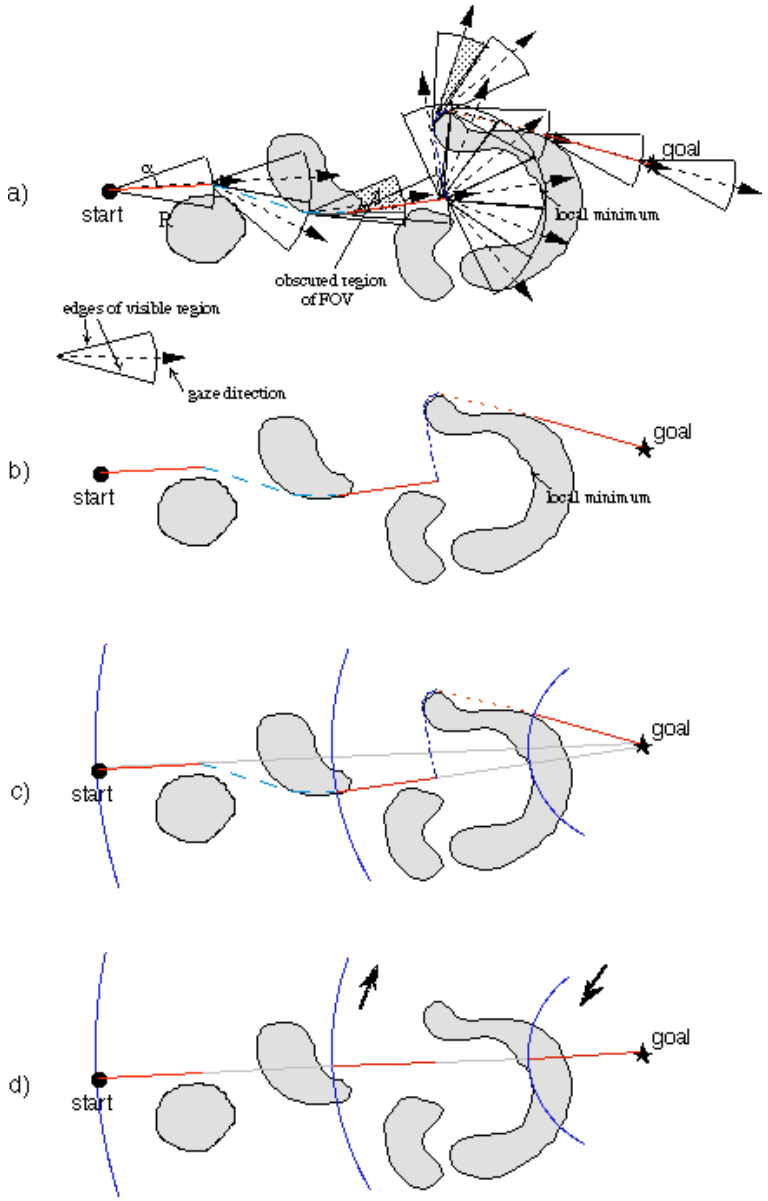


Figure 4.14: Sketch of the proof that the sum of direct segments' lengths is finite. (a) Path generated by Wedgebug, showing the gaze wedges along the way. The second path segment invokes "virtual *MtG*;" the fifth, "virtual *BF*." (To simplify the plot, the wedge toward the goal for each *BF* step is not shown.) (b) The complete path. "Direct" *MtG* segments are solid and dotted. (c) The $\overline{b_i y_i}$ (solid) segments shown as if they are on a nested set of dials centered at the goal. (d) The $\overline{b_i y_i}$ segments' "dials" have been rotated (in the indicated directions) so the segments are aligned. Clearly, the sum of the length of the solid segments is less than the distance from start to goal, and the length of the dotted segment (for the one local minimum encountered) is $\leq R$. (Compare with Figure 3.4 in Chapter 3.)

changed only when a convex corner point x_c of the current generalised critical point is within $G1$. Since the robot executes the entire visible subpath (which passes through x_c) before re-sensing, and since $G1$ contains only those points closer to the goal than the current position, no subsequent $G1$ will contain the same convex corner point. Thus, we have shown that the blocking obstacle may be changed at a finite number of points on each obstacle boundary. □

Lemma 4. *The path length of a single sliding segment, along which the sliding direction and blocking obstacle are fixed, is finite.*

Proof. We first show that the path length D_f followed by the focus point F is finite, then that the path length D traversed by the robot is finite. There are three cases. (a) $[\mathrm{H}_i, \mathrm{D}_i]$: the sliding segment ends when the blocking obstacle changes. Since the sliding direction is fixed, F cannot complete a loop around the blocking obstacle boundary: Cor 4.8 shows that the blocking obstacle is changed (and thus the sliding segment is ended) when F lies on a tangent to the obstacle which passes through the goal. Thus,

$$D_f < \mathrm{meas}(\partial Oi) < \infty.$$

(b) $[\mathrm{H}_i, \mathrm{P}_i]$: the sliding segment ends when a loop is detected. Since the loop is detected by sensing the point V_{loop}, which is explicitly set at H_i as the sensed endpoint of the blocking obstacle in the opposite direction from traversal, then if $F(\mathrm{H}_i)$ is the position of F at H_i, we have

$$D_f = \mathrm{meas}([F(\mathrm{H}_i), \mathrm{P}_i]) \leq \mathrm{meas}([F(\mathrm{H}_i), V_{loop}) \leq \mathrm{meas}(\partial Oi) < \infty.$$

(c) $[\mathrm{H}_i, \mathrm{S}_i]$: the sliding segment ends when the algorithm switches to BF. Since the robot switches to BF before either changing the blocking obstacle (i.e., reaching a tangent point) or detecting a loop, the length of the path followed by F must be bounded by the longest of these two paths:

$$D_f \leq \mathrm{meas}(\partial Oi) < \infty.$$

The motion of the robot during "sliding" segments is a series of steps[†] consisting of straight lines between the current robot position x and $F(x)$. Other than the initial step, during which the robot first acquires the obstacle boundary, the robot skips from point to point on the obstacle boundary (since F is a sensed endpoint of O until the time that sliding segment ends). Thus, these latter steps serve as shortcuts along D_f: for $x \in \partial O$, $\mathrm{d}(x, F(x)) \leq D_f|_{[x,F(x)]}$ (the distance F slides along the obstacle boundary between the initial point x to the step's final focus point position $F(x)$). Clearly, since D_f is finite, and since each sliding step is nonoverlapping and has nonzero path length, D_f is covered by a finite number of steps. Further, since the sliding segment begins when O is first sensed, the initial step must have path length $\leq R$. Therefore,

$$D \leq D_f + R < \infty.$$

□

Lemma 5. *MtG segments are finite length.*

Proof. Consider an entire *MtG* segment, $[\mathrm{L}_i, Y_j]$, where $Y_j = \mathrm{S}_j$ or $Y_j = \mathrm{P}_j$, consisting of several direct and sliding subsegments (we may have $\mathrm{L}_i = \mathrm{H}_{i+1}$). There are a finite number of obstacles, by assumption. Each sliding segment is associated with a single blocking obstacle. By Cor. 4.8, the blocking obstacle can only be changed at a finite number of points on each obstacle. Thus, the total number of possible changes of blocking obstacle is finite. Therefore, there are a finite number of sliding subsegments, which were each shown to have finite length in Lemma 4. Lemma 3 shows that the direct segments have finite length. In conclusion, the entire *MtG* segment, $[\mathrm{L}_i, Y_j]$, has finite path length. □

All of these bounds on path length are conservative, but serve to show that the segments used in Wedgebug are all finite length. All that remains, then, is to show that there are a finite number of these segments.

[†]Recall that a *step* is a single sensing/motion cycle executed by the robot, whereas a *segment* is a contiguous set of steps sharing a common planner mode.

Proposition 5. *The distance from the robot to the goal is nonincreasing during an* MtG *segment.*

Proof. This statement holds trivially due to the definition of $G1$: all nodes V under consideration for the current step must satisfy $\mathrm{d}(V,T) \leq \min(\mathrm{d}(x,T), \mathrm{d}_{\mathrm{LEAVE}})$ (where x is the position of the robot at the start of the current step), for all types of MtG segments. Therefore, if $\overline{x_i y_i}$ is an MtG step, by necessity $\mathrm{d}(y_i,T) \leq \mathrm{d}(x_i,T) = \mathrm{d}(y_{i-1},T) \leq \mathrm{d}(x_{i-1},T)$. (Further, because the step consists of a straight line between x_i and y_i, the robot's distance to the goal along this subpath is nonincreasing.) Since the entire MtG segment $[L_i, Y_j]$ is a connected chain of these steps, we have $\mathrm{d}(Y_j,T) \leq \mathrm{d}(L_i,T)$. □

We note that although the points M_i referenced in the previous proofs are not themselves true local minima in $\mathrm{d}(\cdot,T)$, each M_i is uniquely associated with a (generalised) critical point, m_i, by virtue of the fact that the algorithm switches to BF precisely when the robot is in the basin of attraction of m_i.

Proposition 6. *Wedgebug switches from* MtG *to* BF *when the robot is in a basin of attraction of the distance function, associated with a generalised critical point m_i, on some obstacle boundary ∂O_i.*

Proof. This statement is primarily a restatement of the manner in which Wedgebug uses range sensors to determine when to switch modes. Wedgebug switches from MtG to BF in two cases: 1) when the tangents at both sensed blocking obstacle endpoints point away from the goal, and 2) when the robot is in "sliding mode" and the tangent at the *positive* sensed blocking obstacle endpoint points away from T. In both cases, the robot has detected an inescapable local minimum. We know that the distance function to the goal, $f_T : \mathbb{R}^2 \to \mathbb{R}$ (see Chapter 3) is smooth. Also, since the sensed portion of the obstacle boundary ∂O_i is a compact set, we can consider it a continuous curve, $\varphi : I \to \mathbb{R}^2$. Then, $f_T \circ \varphi$ is continuous, and so attains a maximum and a minimum on I.

Now consider case (1). In this case, the minimum cannot be located at either endpoint, by virtue of the fact that the tangent at each endpoint points away from

the goal—that is, for endpoint e with tangent $\vec{t}_e$ (pointing out from the visible region), $\vec{t}_e \cdot \overrightarrow{eT} < 0$. Since $\overrightarrow{eT}$ is the direction of $-\overrightarrow{\text{grad}}(f_T)$, this orientation indicates that the distance to the goal decreases just inside the visible region from e. Therefore, there exists a local minimum of f_T—and thus a critical point of f_T—within the sensed portion of ∂O_i. Let $\{m_{kx}\}$ be the set of generalised critical points[‡] (i.e., if a critical point is an arc, choose one point to represent that basin) sensed from robot position x. Let m_i be the sensed g.c.p. closest to T (thus, we have $\mathrm{d}_{reach} = \mathrm{d}(m_i, T)$ when the robot is at x).

For case (2), note that the robot has been monotonically decreasing its distance to T as it has travelled along the obstacle boundary. Thus, the robot will need to monotonically increase its distance from T if it backtracks around the obstacle boundary. Thus, we have the same situation as in (1), unless the last portion of the obstacle boundary traversed was a circular arc with its center of curvature at T. In this case, the arc is a g.c.p. Z, so the negative sensed obstacle boundary is a critical point of the distance to T. Again, let $\{m_{kx}\}$ be the set of g.c.p.'s sensed from x, and choose m_i to be the sensed g.c.p. closest to T. □

For convenience, we will say that Wedgebug switches modes "at" local minimum m_i.

Lemma 6. *The distance from T to successive local minimum points m_{i-1}, m_i (associated with BF switch points) is decreasing:* $\mathrm{d}(m_i, T) < \mathrm{d}(m_{i-1}, T)$.

Proof. There are two cases, given *BF* segment $\overline{\mathrm{S}_i Y}$, where $Y = L_i$ or P_i: a) $\overline{\mathrm{S}_i Y}$ directly follows a *BF* segment. Since the algorithm did not halt after the prior *BF* segment–that is, the leaving condition was met–the prior segment must be of type $[\mathrm{S}_{i-1}, \mathrm{L}_{i-1}]$. Further, since the end of any *BF* segment triggers *MtG*, we must have $\mathrm{S}_i = \mathrm{L}_{i-1}$ (otherwise, an *MtG* segment would intervene). At L_{i-1}, we know that there exists a node V in $G1(\mathrm{L}_{i-1})$ which satisfies the leaving condition $\mathrm{d}(V, T) < \mathrm{d}_{reach}$, by definition, where $\mathrm{d}_{reach} \leq \mathrm{d}(m_{i-1}, T)$. Since the algorithm switched

[‡]Recall that we define a *generalised critical point* (g.c.p.) Z of the distance to T as either an isolated critical point (including "vertices" in the obstacle boundary) or as a path-connected "arc" of nonisolated critical points. (See Section 3.4.3 for an exact definition.)

immediately to *BF*, $\overline{W}(\mathrm{L}_{i-1})$ must contain the local minimum m_i. Because m_i is a local minimum in $\mathrm{d}(\cdot, T)$, we must have $\mathrm{d}(m_i, T) \leq \mathrm{d}(V, T)$. Therefore, we have

$$\mathrm{d}(m_i, T) < \mathrm{d}_{reach} \leq \mathrm{d}(m_{i-1}, T).$$

b) The *BF* segment follows an *MtG* segment. In this case, if X is the subpath goal of the initial step of the *MtG* segment, and S_i is the switch point marking the start of the *BF* segment, we have $\mathrm{d}(\mathrm{S}_i, T) \leq \mathrm{d}(X, T)$ by Proposition 5. The prior *BF* segment, as in (a), is of form $[\mathrm{S}_{i-1}, \mathrm{L}_{i-1}]$. Again, there exists a node V in $G1(\mathrm{L}_{i-1})$ which satisfies the leaving condition $\mathrm{d}(V, T) < \mathrm{d}_{reach}$, where $\mathrm{d}_{reach} \leq \mathrm{d}(m_{i-1}, T)$. Before the start of the intervening *MtG* segment, $\mathrm{d}_{\mathrm{LEAVE}}$ is reset to $\mathrm{d}(V, T)$. Thus, $\mathrm{d}(X, T) \leq \mathrm{d}_{\mathrm{LEAVE}} = \mathrm{d}(V, T) < \mathrm{d}_{reach} \leq \mathrm{d}(m_{i-1}, T)$. Finally,

$$\mathrm{d}(m_i, T) \leq \mathrm{d}(S_i, T) < \mathrm{d}(m_{i-1}, T).$$

□

Now we can show the following result:

Corollary 2. *Wedgebug will switch from MtG to BF at most once for a given basin of attraction.*

Proof. This statement is a consequence of the fact that the distance to T decreases between successive (generalised) local minima of f_T which trigger mode switches along the path, and that the distance to T is constant for all points in the ω-limit set of a given basin. □

Lemma 7. *Wedgebug terminates after a finite path.*

Proof. There are a finite number of obstacles, each with finite boundary. By assumption, $\mathrm{d}(S, T) < \infty$. *MtG* can switch to *BF* only at a local minimum, m_i, of $\mathrm{d}(\cdot, T)$. By Lemma 6, each of these switches occurs closer to the goal than the prior switch, so the behaviour can switch only once at each local minimum. In addition, Lemma 1 from Chapter 3 tells us that there are a finite number of relevant critical

points (basins) in the bounded area $B_T(\mathrm{d}(S,T)+R)$. Since each m_i is a critical point, and since each *MtG* segment is constrained to start within $B_T(\mathrm{d}(S,T)+R)$ by the parameter $\mathrm{d}_{\mathrm{LEAVE}}$–and thus all m_i encountered lie within $B_T(\mathrm{d}(S,T)+R)$– there are a finite number of switches between behaviours. Each *BF* segment ends either at a loop, or when the leaving condition is met, when the behaviour switches back to *MtG*. Each *MtG* segment (here we mean a complete *MtG* segment, including both direct and sliding portions) ends either at a switching point (associated with a unique critical point), at a loop, or at the target. Thus, there are a finite number of segments of each type. By Lemma 2 and Lemma 5, each segment ends after finite path, so the total path has finite length. □

Lemma 8. *Wedgebug is complete.*

Proof. Lemma 7 shows that Wedgebug completes after finite path length. All that remains is to show that, if the goal is reachable from the initial robot position, the algorithm will reach it. This is accomplished by a similar argument to Theorem 2 from [21] (see Chapter 3), which uses the fact that there are a finite number of critical points (basins) to demonstrate that (if T does not lie on some ∂O, causing the path to terminate after a *BF* segment) there must necessarily be a last *MtG* segment, terminating at the goal. □

Lemma 9. *An upper bound on the total path length generated by the Wedgebug algorithm is*

$$\mathrm{d}(S,T) + R \times \sum_j \big(K \times (\#Minima)_j + (\#Tangents)_j + \frac{1}{R}\,\mathrm{meas}(\partial O_j) + 1\big),$$

where: j *ranges over all obstacles intersecting the disc* $B_T(\mathrm{d}(S,T))$,
$\mathrm{meas}(\partial O_j)$ *is the perimeter of obstacle* O_j,
$K = 3 + \mathrm{meas}(\partial O_j)$,
$(\#Minima)_j$ *is the number of local minima of* $\mathrm{d}(\cdot,T)$ *around the perimeter of obstacle* O_j,
$(\#Tangents)_j$ *is the number of points on* ∂O_j *tangent to a line*

through T.

Proof. Lemma 3 yields

$$D^{direct} = R \times (\#Minima) + \mathrm{d}(S,T)$$

as the bound on the sum of the path lengths of all "direct" MtG segments, where $\#Minima$ is the total number of local minima in $\mathrm{d}(\cdot,T)$ within $B_T(\mathrm{d}(S,T))$.

Lemma 2 gives the bound on path length for each BF segment as $R+\mathrm{meas}(\partial O_i)$, where O_i is the followed obstacle. The sum of the path lengths of all BF segments is bounded by $\sum_i(\mathrm{meas}(\partial O_i)+R)$, where i ranges over the followed obstacles. In particular, since each BF segment is associated with a single local minimum in $\mathrm{d}(\cdot,T)$, then for a single obstacle O_i the sum of the path lengths for BF segments following O_i is bounded by $(\mathrm{meas}(\partial O_i)+R)\times(\#Minima)_i$, where $(\#Minima)_i$ is the number of local minima of $\mathrm{d}(\cdot,T)$ around the perimeter of O_i. Therefore, the sum of the path lengths of all BF segments can be bounded by

$$D^{\mathsf{BF}} = \sum_j((\mathrm{meas}(\partial O_j)+R)\times(\#Minima)_j),$$

where j ranges over all obstacles intersecting $B_T(\mathrm{d}(S,T))$.

Since the distance to the goal is nonincreasing during MtG segments (Proposition 5), and the distance between MtG segments is decreasing (by Lemma 6), the robot cannot traverse the same portion of a given obstacle's boundary twice during two different "sliding" MtG segments. Thus, the path length of the focus point F over all "sliding" subsegments can be bounded by $D_f^{total} \le \sum_j \mathrm{meas}(\partial O_j)$, where j ranges over all obstacles intersecting $B_T(\mathrm{d}(S,T))$. Now, Lemma 4 shows that each "sliding" segment adds a length R for the robot's path. Thus, we need to bound the number of "sliding" segments. This number is bounded by the number of ways a "sliding" segment can be terminated: by changing blocking obstacle, switching to BF, or detecting a loop. Thus, for a single obstacle O_i, the bound on the number of all "sliding" MtG segments following that obstacle's boundary is $(\#Tangents)_i +$

$(\#Minima)_i + 1$, where $(\#Tangents)_i$ is the number of points on ∂O_i tangent to a line through T, and $(\#Minima)_i$ is the number of local minima of $\mathrm{d}(\cdot, T)$ around the perimeter of O_i. Therefore, the sum of the path lengths for all "sliding" *MtG* segments is

$$D^{slide} = \sum_k (R + D_f^k),$$

where k ranges over all "sliding" segments, and

$$D^{slide} = \sum_k R + \sum_k D_f^k \leq \sum_k R + \sum_j \mathrm{meas}(\partial O_j),$$

where j ranges over all obstacles intersecting $B_T(\mathrm{d}(S,T))$. Finally, we have

$$D^{slide} \leq R \times \sum_j ((\#Tangents)_j + (\#Minima)_j + 1) + \sum_j \mathrm{meas}(\partial O_j).$$

To summarise, the bound on the total path length is then

$$\begin{aligned} D^{total} &= D^{direct} + D^{BF} + D^{slide} \\ &\leq \mathrm{d}(S,T) + R \times (\#Minima) + \sum_j ((\mathrm{meas}(\partial O_j) + R) \times (\#Minima)_j) \\ &\quad + R \times \sum_j ((\#Tangents)_j + (\#Minima)_j + 1) + \sum_j \mathrm{meas}(\partial O_j) \\ &\leq \mathrm{d}(S,T) + R \times \sum_j \big((3 + \mathrm{meas}(\partial O_j))(\#Minima)_j + (\#Tangents)_j \\ &\quad + \frac{1}{R} \mathrm{meas}(\partial O_j) + 1 \big). \end{aligned}$$

□

4.6 Summary and Discussion

In this chapter, we have presented the Wedgebug algorithm, a sensor-based motion planner designed for robots with a sensor FOV limited in both downrange and angular scope. The algorithm uses a simple local world model, and invokes automatic

gaze control, based upon the sensed environment, to minimise sensing, to avoid unnecessary robot motion, and to improve overall efficiency. We have also presented the proof that the Wedgebug algorithm is complete and correct, and have provided an upper bound on the resulting path length.

During the creation of the Wedgebug algorithm, we made several choices based upon tradeoffs between robot motion and sensing. The resulting algorithm is complete, and produces paths which are the shortest possible given the knowledge gained from only the regions sensed. However, it turns out that the final paths produced by Wedgebug are sensitive to the initial conditions, and to noise in measurements (especially noise in calculation of obstacle tangents). That is, slight perturbations can change the direction the robot chooses to skirt around a given obstacle, for instance. The algorithm presented here strives, perhaps to an extreme, to minimise the number of sensor queries: once a suitable subpath goal is found, the robot moves and begins the next step. Several variations to the basic Wedgebug methodology can be used to improve global performance (i.e., come closer to–or achieve–true local optimality and improve the deterministic quality of the results), at the cost of added sensing. For example, the robot could sense, at each step, the full extent of the blocking obstacle boundary, using as many wedge views as necessary. An intermediate choice would have the rover sense wedges until it has enough information to make a well-informed decision (once it has found a suitable subpath goal, say for "virtual BF," it may sense one more wedge in the opposite direction just to be certain an equal or better potential goal does not appear). All three variants' performance could be enhanced by sensing more often–that is, by re-sensing before the subpath goal is reached, one or more times (in this case, we need to add a condition similar to the *detour condition* in [21] to prevent oscillations in the path). Most importantly, since the extreme version of the algorithm presented here (as well as TangentBug, which is essentially the extreme case where all possible wedges are constantly re-sensed) is complete, it is clear that carefully constructed variations such as those mentioned above will also retain this crucial property.

Chapter 5

Implementation and Results

5.1 Motivation Redux

As has been discussed in prior chapters, the primary motivation for the work described herein is the upcoming series of missions planned for Mars exploration, with launch dates every two years from 1998 through 2005, culminating in samples returned to Earth from the Red Planet in 2008. Three of these missions feature rovers, independent mobile spacecraft which will roam the planetary surface in order to conduct *in situ* experiments on a variety of samples, in various terrains spread over regions potentially kilometers square. These "mobile geologists," as they have been called, will not only learn more about martian geology and geochemistry, etc., but also determine which particular samples should be sealed inside a capsule bound for Earth, where more extensive study is possible than on Mars. These rovers are direct descendents of the Sojourner rover, which landed on Mars as part of the Mars Pathfinder mission in July, 1997. In order to expand upon this spectacularly successful technology demonstration, the future planned Mars rover missions* require mobile robots which are capable of being operated for up to a year, traversing much larger distances (up to 100 m/sol, as opposed to Sojourner's 102 m/83 sols), carrying more instruments to a wider variety of features, and caching samples along the

*with the exception of the 2001 mission, which, due to technical and political constraints, will fly a virtual copy of the Sojourner rover. In fact, the 2001 rover will be the refurbished flight spare from the Pathfinder mission, named "Marie Curie."

way [53].

A key advance in functionality required for these new planetary rovers is greater navigational autonomy, since longer distances must be covered between opportunities to communicate with Earth than in prior missions. For example, the rover mission planned for 2003 (carrying science payload "Athena") would have the rover traverse up to 50-100 meters per sol, between approximately 10AM to 3PM Mars local time. However, the rover will only have limited opportunities to communicate to operators on Earth and to receive instructions. Furthermore, it will not be possible to anticipate (and thus pre-program) the proper reactions of the rover to the terrains encountered during the traverse: each rover will be working in unknown, rough terrain. (The resolution expected from Mars orbiters, for example, is roughly 300 meters/pixel, with only isolated "postage stamp" regions achieving the highest resolution of 1.4 m/pixel [50]. Orbiter camera pointing limitations prohibit attempting to use these highest-resolution images for rover navigation or localisation.) Thus, an efficient, on-board planner is needed to ensure the rover will make acceptable progress toward each goal, and to achieve each goal accurately, during long stretches of unsupervised time during each sol.

As discussed in Chapter 4, useful motion planners for planetary rovers must satisfy several prerequisites, among which are severe constraints of limited power and computational capacity, and the high cost of flight components, which translates into limited memory available on-board the rover. For example, in the 2003 mission described in Chapter 2, the rover will use a rad-hard R3000 10 MIPS processor and will have 1-6 MB of on-board memory, of which perhaps 0.5 MB can be used by the motion planner during operation. One ramification of the limitation on computational capacity is the time required for executing the motion planner. If the planner contains heavy computation which, for example, must be executed after every 10 meter step, a lengthy traverse will take much more time than a traverse utilising a planner which requires less processing. Not only does this fact impact the length of the possible traverse (due to time-of-day restrictions based on available solar power), but also how much of that sol's duration can be dedicated to science.

Increasing the proportion of "science time" during each sol is crucial, particularly when bearing in mind the necessarily limited lifetime of the robotic vehicle.[†] Thus, a practical motion planner must utilise the available sensing array in a scheme which efficiently senses only the data needed for motion planning, requires minimal memory to store salient features of the environment, and conserves rover motion.

The crux for developing a practical motion planner for the next several planned missions to Mars is to be able to implement the motion planner on a flight-like testbed, to ensure both the correct operation of the algorithm in a real-world scenario, but also to be certain the algorithm fits well within the constraints of computational capacity, required memory, and time. The remainder of this chapter describes the implementation of the Wedgebug algorithm (described in detail in Chapter 4) on the Rocky7 prototype microrover at the Jet Propulsion Laboratory (JPL). This system is operated primarily in JPL's MarsYard. Section 5.2 describes the testbed vehicle and environment, while Section 5.3 discusses the specific difficulties posed by the testbed and the changes to the algorithm necessary to meet these challenges. The next section details the actual implementation of the resulting "RoverBug"[‡] algorithm, including both the expected operational scenario and the relevant portion of the actual navigation software system for the Rocky7 rover. Finally, we present the results from experimental traverses in the MarsYard and elsewhere in Section 5.6.

5.2 Description of the Testbed

The Jet Propulsion Laboratory has produced several robotic vehicles, serving various purposes. Most germane to the topic at hand is the series of "microrovers," small-size six-wheeled robots featuring rocker-bogie suspensions, destined either to fly to Mars or to remain Earth-bound and test out future technologies for their far-reaching siblings. Of the latter tribe is the Rocky7 prototype microrover, developed

[†]There is currently no possibility of repair on Mars.

[‡]so named due to its descendance from the Wedgebug algorithm (and, in turn, from TangentBug and the original "Bug" algorithms), as well as for being tuned specifically for the challenges raised by an actual rover

by the Long Range Science Rover task at JPL and pictured in Fig. 5.1 [82].

Rocky7 is the next vehicle in the series of research rovers which spawned the Sojourner rover (Sojourner is the direct descendant of Rocky 4).[§] He is roughly the same size as Sojourner, standing 60 cm long by 50 cm wide by 35 cm tall, and weighing 15.7 kg. (By comparison, Sojourner is 68 cm x 48 cm x 28 cm and 10.5 kg, and the concept prototype 2003 rover, FIDO[¶], is roughly 1.5x larger in every dimension, weighing in at 37 kg (including instrument payload) and standing 100 cm x 75 cm x 45 cm tall [54],[11].) As mentioned above, Rocky7 features a six-wheeled rocker-bogie suspension. This type of chassis enables the rover to surmount obstacles 1.5 wheel diameters tall[‖] while the electronics box in the vehicle's center experiences only the averages of the suspension's contortions, remaining mostly level. The suspension consists of five passively pivoting links, assembled in the following manner: The *bogie* connects the rear[**] pair of wheels on each side. The *rocker* on each side couples the bogie with the "free"—and in this case, steering—wheel, and

Figure 5.1: The Rocky7 Prototype Microrover, developed at JPL to test technologies for future missions. He is pictured here with his mast raised in the JPL MarsYard, an outdoor testing arena featuring simulated martian terrain.

[§]Rockies 5 and 6 were never built, but their names serve as placemarkers for how their designs fit into microrover evolution.

[¶]Field Integrated Design & Operations

[‖]Rocky7's wheels are Sojourner spares and are 13 cm across; the 2003 rover is expected to have at least 20 cm diameter wheels.

[**]It may appear in the image of Rocky7 in Fig. 5.1 that the coupled pair of wheels are on the *front* of the vehicle, when in fact the "arm" side of Rocky7 is defined as the *rear*. During testing, we even drive Rocky7 in this direction, "backward," 99% of the time. Note that this coordinate frame definition is reversed from Sojourner's body frame.

A Closer Look at the Rocker-Bogie Mechanism

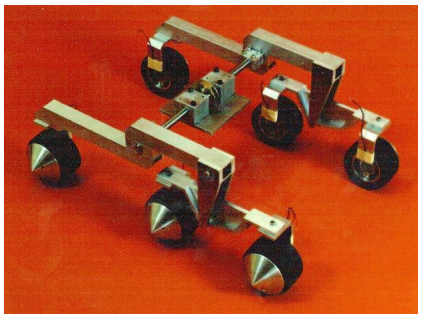


Figure 5.2: Photo of Rocky 1. This demonstration robot consists solely of the rocker-bogie mechanism, sans differential. The bogies are to the right in this image.

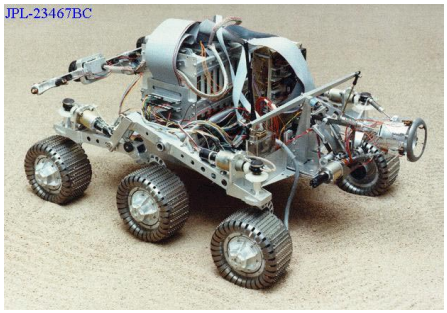


Figure 5.3: Photo of Rocky 4.2. This software development model shows the differential (right, behind the cylindrical APXS mockup). The bogies are to the left in this image.

(both images courtesy JPL)

pivots on a "jeff tube"[††] which runs through the chassis box and joins the two sides. Finally, the *differential* connects the free wheels' end of the rockers and is mounted on the front of the chassis box. (See Figures 5.2, 5.3.) The rover's top speed is roughly 1.7 cm/sec. Rocky7's other, un-Sojourner-like features are prompted by the requirements for future missions. The rover boasts three stereo pairs of cameras for navigation—two body-mounted front and rear, and one on a deployable 1.2 m, 3 degree of freedom (DOF) mast—as opposed to Sojourner's body-mounted cameras and laser striping system. (The mast can only be fully deployed when the rover is stationary, however. The mast has an intermediate raised position which it can hold while the rover is in motion, with the drawback that in this position, the mast is unable to pan, and can tilt only within a reduced range.) Besides a flight-like "gyro"[‡‡] and odometry sensors, Rocky7 also uses a photovoltaic cell-based sun sensor for absolute heading measurement. The 2003 rover is planned to feature a permanently-deployed mast carrying a wide-baseline stereo pair of cameras for navigation (as well as various bore-sighted instruments), and will most likely also use some type of sun sensor for determining her absolute heading. To complete the

[††]This is simply a tube which connects the two sides of the rover's rockers, incidentally used to hold the RHU's for Sojourner. Why "jeff tube?" Because, when a JPL engineer was told to come up with a name for it, he replied, "Oh, you mean like 'Jeff'?"

[‡‡]actually, a turn rate sensor

	Rocky7	Sojourner	2003 rover
size	60 cm x 50 cm x 35 cm	68 cm x 48 cm x 28 cm	145 cm x 120 cm x 60 cm
mass	15.7 kg	10.5 kg	70 kg
wheel	13 cm dia.	13 cm dia.	20 cm dia.
speed	1.7 cm/sec	0.7 cm/sec	6 cm/sec?
CPU	68060 @ 100 MIPS	80C85 @ 100 KIPS	R3000 @ 10 MIPS
mem	16 MB	752 KB	6 MB
OS	VxWorks®	none	VxWorks®
navig cams	mast stereo pair 2 body stereo pairs	body stereo pair laser striping system	mast stereo pair 2 body stereo pairs
cam FOV	mast: $43.7° \times 33.7°$ body: $103.0° \times 94.2°$	body: $127.5° \times 94.5°$	mast: 45°? body: 120°?
cam res†	mast: 512×480 body: 384×240	body: 768×484	mast: 512×512? body: 512×512?
base-line	mast: 10 cm body: 5 cm	body: 12.6 cm	mast: 15 cm body: 10 cm

Table 5.1: Comparison of Rocky7 with Sojourner and proposed 2003 rover

list of features, though not relevant for navigation, Rocky7 also carries a short, 2 DOF arm with clamshell scoops and an integrated fiber optic cable for an internally-mounted spectrometer, and can carry an additional instrument on the end of his mast, for placement on a rock or on soil in back of the rover.

For computation, Rocky7 is fitted with a 68060 CPU in a 3U VME chassis, running at approximately 100 MIPS, and features 16 MB of memory. (Continuing the comparison, Sojourner used a 80C85 CPU at 100 KIPS, with 752 KB total memory; the 2003 rover will use an R3000 at 10 MIPS, and likely will have 6 MB of memory. The increased memory and computational power on-board Rocky7, while still providing flightlike constraints, allows for more speedy testing and timely implementation of non-optimised experimental code.) The majority of the electronics, motors, and sensors are COTS* components, with a minimum of custom adjustment. Rocky7's operating system is Wind River's VxWorks®, and his on-board software

*Commercial Off-The-Shelf

†The table shows the full resolution of the cameras. Vision processing, however, may be done at a different resolution. On Rocky7, for example, obstacle detection is done at a resolution of 128×120 for the mast cameras, and 96×60 for the nav cameras, primarily to speed processing time. These resolutions represent two levels removed from full resolution

is written in C and C^{++}, in part using Real Time Innovation's ControlShell® environment. ControlShell provides a graphical interface for designing interacting finite state machines and data flow block diagrams for a real time system, and manages a database and program execution during operations. In addition to these on-board faculties, the rover is operated using a suite of off-board ("ground-based") software, including WITS[‡], developed by Paul Backes at JPL, as our rover control workstation; SCE[§], developed by Steve Peters at JPL, as the executive; and SCE-forward, also by Steve Peters, to shuttle commands and telemetry between the rover and WITS/SCE. The ground-based software is written in Java and Lisp.[¶]

To round out Rocky7's hardware system description, the rover is powered by rechargeable NiCad batteries (4 strings of 4 batteries each), aided by a Si solar panel, for a yield of about 35W on average. (By comparison, Sojourner's primary power source was a GaAs solar panel, aided by non-rechargeable Li thionyl chloride batteries, yielding roughly 16W peak. The precise power budget for the 2003 rover has not yet been finalised, but her primary power source will be a solar panel, as well as rechargeable Li ion batteries.) Unlike the true Mars rovers, Rocky7's environmental conditioning strives to keep the rover cool in hot desert climates, not warm against Mars' deep chill. Thus, rather than boasting aerogel insulation and heaters, the rover's electronics chassis is shaded by a raised solar panel and cooled by nine chassis-top fans, which together keep the electronics box temperature below 70° F. The rover communicates with his ground station via either an ethernet/serial cable, or a wireless ethernet system, depending upon the requirements of the test.

The vehicle, as a testbed for technologies for future missions, carries an evolving set of on-board software, developed by the Long Range Science Rover Task. At the time of writing, the vast majority of the software resided within the aforementioned ControlShell environment. This software includes everything from device drivers to

in a pyramid processing scheme; each level (reducing resolution by a factor of 2 in each dimension) reduces processing time by roughly a factor of 8 [46].

[‡]Web Interface for Telecience

[§]Sequencing and Command Executive

[¶]SCE and SCE-forward, originally written in Lisp, have now been ported to C^{++} and will soon reside on-board the rover.

high level functionality. Key to the work presented here are several pieces of vision processing code, which produced range images[‖] from full stereo. The algorithms were originally developed by Larry Matthies and coded by Todd Litwin; the versions used on-board Rocky7 were adapted by Bob Balaram [49],[48]. In brief, given a stereo image pair, the image processor measures image similarity via correlation windows within a fixed disparity search range, then estimates the disparity for each pixel independently. The disparity estimate is refined using a parabolic fit to estimate the subpixel disparity. Finally, the disparity map is smoothed, and each pixel's final disparity value is used to produce the pixel's X-Y-Z coordinates via triangulation. A second processing step detects positive[**] obstacles within the resulting rangemap. Another piece of code which proved invaluable was Clark Olson's localisation algorithm [59]. This function allows the rover to estimate his motion by scanning a given target point with the mast-mounted cameras, both prior to and after a short traverse, then comparing the binned elevation maps using a best-fit Hausdorff measure. Since Sojourner's measured error in dead-reckoning was on the order of 5-10% of distance travelled for each traverse, the periodic updates to the rover's estimation of his position available from the localisation algorithm greatly increase confidence in the rover's ability to reach his goal, assuming an appropriate navigator is available.

5.3 Challenges for Wedgebug

There are several pieces to such an "appropriate navigator." At the highest level is the grand planner, charting the general regions for traversal and taking into account obstacles or otherwise undesireable regions on a grand scale. For the class of microrovers with which we are concerned, these obstacles/terrain features are on the order of > 500 m in length, roughly. The typical mission scenario calls for the grand planner to be Earth-based, customarily a team of scientists and rover opera-

[‖] We here use "range image" and rangemap interchangeably.

[**] As described in [48]: "Positive obstacles are those that extend upward from the nominal ground plane, like rocks, bushes, and fence posts. 'Negative' obstacles extend downward, like potholes, man-made ditches, and natural ravines."

tors/engineers, working from orbital or descent imagery. Next is a high-level path planner, which is capable of charting a path through rough terrain, to the regions or waypoints designated by the grand planner. For Sojourner, this level was off-board as well, consisting of the team of "Rover Drivers" at JPL. However, for future missions, since the rover will be expected to traverse much longer distances between communication opportunities with Earth, this functionality must be migrated on-board the rover. This level—the autonomous on-board path planner—is the focus of the work presented here. Finally, the lowest level is the piloting algorithm, which executes the path generated by the path planner. This level contains the control laws for actual rover motion, as well as ideally some measure of fault tolerance, such as a hazard avoidance scheme in case obstacles were missed by the path planner's sensors. Rocky7's piloting algorithm, like Sojourner's, consists of two switchable behaviours: a "go direct" mode, which executes the given commands without explicit obstacle sensing; and the "go to waypoint" mode, which includes several heuristic sub-behaviours to avoid detected obstacles and seek the given waypoint.[††] Work is currently underway to develop a better piloting algorithm for Rocky7, capable of intelligently servoing on a path or a string of waypoints while avoiding obstacles.

We thus turn our attention to the challenges presented for developing and implementing a practical high-level path planner for flight-class planetary microrovers, and Rocky7 in particular. Although the Wedgebug algorithm described in Chapter 4 is an important step toward a practical planner, it still does not quite capture the complexities of the real world. For instance:

- The real rover is not a point robot with omnidirectional motion capabilities, and it moves in "2 1/2" dimensions over rough terrain rather than the theoretical 2. The real size of the rover is addressed in the "RoverBug" implementation (see Section 5.4) by calculating the obstacles' "silhouettes": the smallest polygon bounding the projection of each $SE(2)$ obstacle onto $\Re^2$.

[††]The names for these two modes are not official; Sojourner did not have a formal "go direct" mode (instead, explicit "move" or "turn" commands were given), and Rocky7's analogue to Sojourner's "go to waypoint" command is the command, "go via."

- Rovers have notoriously poor dead reckoning abilities [54], yet the algorithm requires the rover to maintain accurate positioning to recognise both the goal position and when an obstacle has been circumnavigated. (Sojourner was unable to maintain an accurate model of her position within a global coordinate frame.) The localisation algorithm mentioned in Section 5.2 helps maintain Rocky7's knowledge of his position.

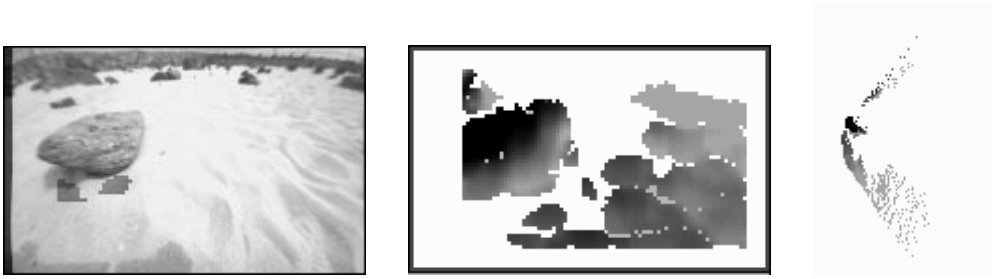
- The mast imagery can "see over" many obstacles: the martian terrain so far encountered has featured fields of rover-size rocks. These obstacles block motion, but not necessarily sensing, which adds complexity beyond that captured by the "wall model" of typical theoretical obstacles.

- Rocky7's mast is limited (in the current implementation of its obstacle detection software) in its ability to sense obstacles within 1-1.5 m of the rover. The obstacle detection algorithm (described in Section 5.5) essentially searches for steps in elevation, which are not easy to detect while looking straight down on the tops of rocks. As a result, the *BF* mode of Wedgebug has been modified to make maximum use of the near-sighted body cameras.

- Wedgebug assumes an environment with "binary obstacles"—that is, areas are labelled as traversable or forbidden, with no middle ground or continuum of values. Although this limitation is not specifically addressed by RoverBug, it is possible to define "risk levels" which can be designated by ground-based operators, and then used to tune which areas are deemed too risky for traversal at that juncture.

5.4 RoverBug

In response to these challenges, we have developed the "RoverBug" algorithm, a version of the Wedgebug algorithm which has been tuned and modified for the Rocky7 vehicle described in Section 5.2. In this section, we discuss the most significant changes made during the migration from Wedgebug to RoverBug.



(a) (b) (c)

Figure 5.4: Sample image and range data from body-mounted stereo pair. (a) Image from left camera of stereo pair. (b) Elevation map of pixels (in greyscale), determined using stereo triangulation (white pixels indicate no data). (c) Overhead view of elevation map (rover is at left). The pixels marked in the image (which correspond to the black "blob" in the overhead plot) show where an obstacle was detected.

5.4.1 The Eyes Have It

The primary driver behind many of the adaptations required for the RoverBug implementation is the nature of Rocky7's sensor suite. For purposes of path planning, the most useful sensors on board the rover are the cameras, which produce stereo-derived rangemaps of the robot's surroundings, and a sunsensor, which returns the rover's absolute heading. As previously discussed, a localisation algorithm uses the cameras to return odometry-like information. Of these sensors, arguably the most important (and most idiosyncratic) are the cameras. Rocky7 is fitted with three stereo pairs of cameras, each with different properties, image footprints, and utility. The two pairs of body- (or chassis-) mounted cameras, called the "nav" cameras, are fixed in place, unable to be panned or tilted. They each have roughly a $103^\circ \times 94^\circ$ field of view (FOV)[‡‡] with a 5cm baseline, and are mounted on the front and rear of the vehicle just under the solar panel, 0.31m above the ground and tilted 40° down from the horizontal(see Fig 5.4 for example data). The mast stereo pair, on the other hand, can be panned and tilted (by gross motion of the mast shoulder and elbow joints). The mast cameras have a $43.7^\circ \times 33.7^\circ$ FOV, are separated by 10cm, and are mounted 1.42m above the ground (see Fig 5.5 for example data). Although

[‡‡]However, stereo range data can only be produced within an 80° (horizontal) FOV, due to the fisheye lenses on the body-mounted cameras.

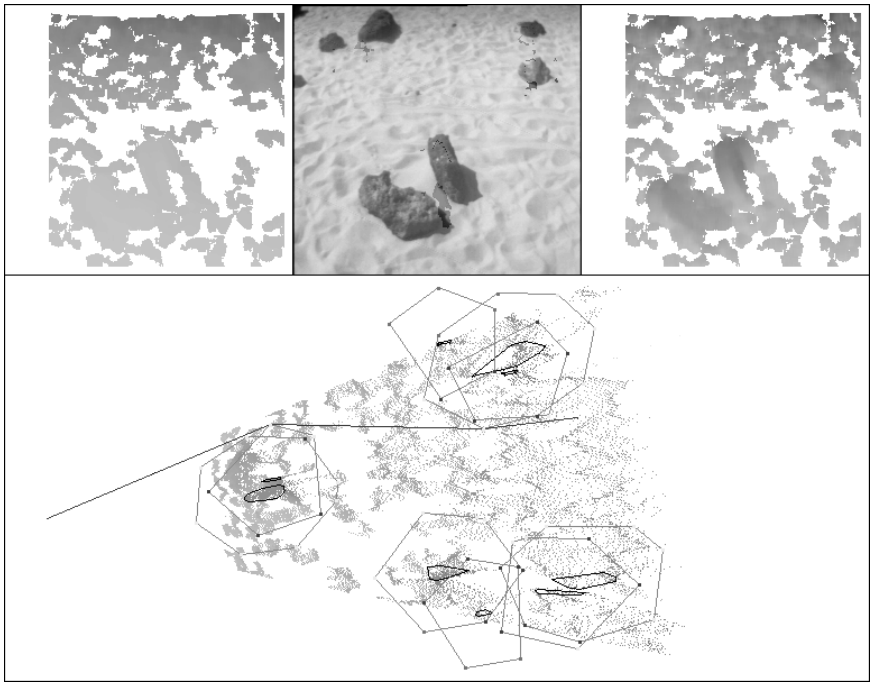


Figure 5.5: Sample image and range data from mast-mounted stereo pair. The middle image is from the left camera. The upper left and right plots depict the depth and elevation maps, respectively, and the lower plot is an overhead view of the elevation data. The pixels marked in the image show where obstacles were detected. Also shown are the obstacle convex hulls and silhouettes, as well as a path computed through the wedge.

Rocky7's mast-based stereo pair provides a higher vantage point for imagery than the body-mounted cameras, and thus potentially allows more distance to be covered between sensor queries, the footprint of the mast cameras is not a perfect wedge, with the rover at the apex. Although the mast can certainly image regions close to the rover, the limitations of the Rocky7 implementation of the obstacle detection algorithm used (described in [45], [48]) do not allow for robust obstacle detection within roughly 1.5 m of the rover.

A choice must be made whether to use a single mast "wedge view" for path planning, or whether to let a single "wedge" be a composite of contiguous views taken using a set pattern of pan and tilt angles (see Fig. 5.6 for an example of a mast multi-view "wedge"), or even to use a combination of mast and nav camera views. The latter option provides information about obstacles directly in front of the

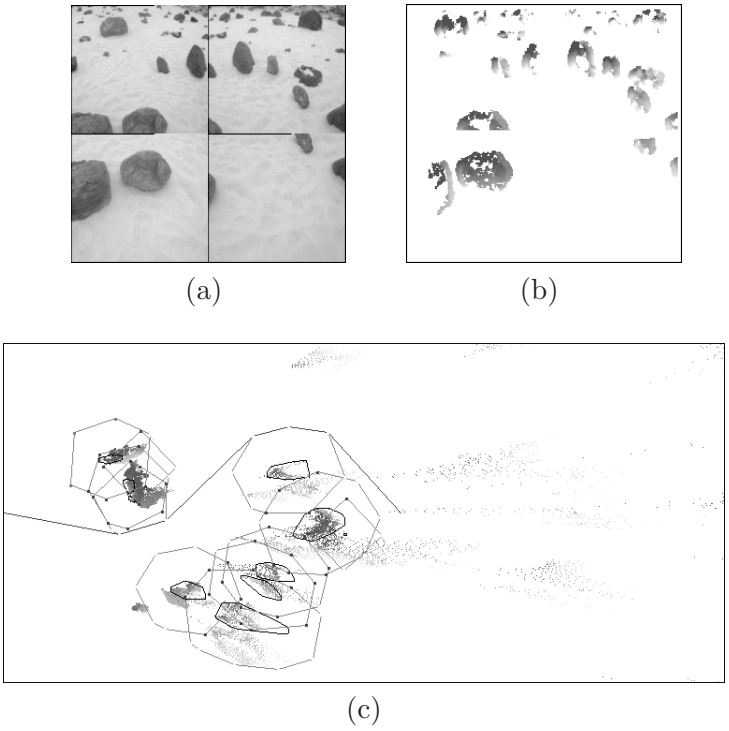


Figure 5.6: Results from a multi-image "wedge" view. (a) Left images from the mast-mounted stereo pair (b) Pixel elevation map (greyscale) determined using stereo triangulation (white pixels indicate no data) (c) Overhead view of elevation map, with detected obstacles' convex hulls and corresponding silhouettes, and a path computed through the data from the four combined views

rover, missing in the other options. However, it must be emphasized that since a nav camera view cannot be changed without moving the rover, this option is generally unavailable for the "virtual" modes of Wedgebug. (In addition, it should be noted that the Rocky7 rover's lack of side-viewing proximity sensors of any kind greatly impacts our implementation of the boundary following mode.) We have chosen to implement the capability of utilising multiple views as a single wedge, including adding a forward-looking view from the appropriate chassis-mounted stereo pair. However, in the interest of demonstrating the ability of the path planner to function using a minimum number of sensed views, in general we have defined a "wedge" to be a single mast view, plus a single chassis-camera view. (Wedges in any direction other than straight towards the goal consist solely of a single mast view.)

5.4.2 High Vantage Advantage

Using the mast cameras for path planning yields the advantage of being able to "see over" many of the surrounding rocks (see Fig. 5.5). As a result, rather than being limited to the simple "star-shaped graph"—consisting of edges radiating out from the rover's position x to the sensed obstacle boundary endpoints—used by Wedgebug, the RoverBug algorithm's LTG is truly the portion of the tangent graph limited to the visible wedge (see Fig. 5.7). Therefore, RoverBug uses an A* algorithm [63] to search this richer LTG for the locally optimal (shortest length, considering only the visible obstacles) path to T.

Since the planner now has a model of the terrain within the entire wedge (instead of a single range contour incorporating only the closest obstacles), the subpath should no longer be generated by simply returning the first "leg" of the computed path (i.e., the first LTG node encountered, defined as the "focus point" for Wedgebug). Instead, we produce a subpath by truncating the shortest path to T at the edge of the visible region, along the arc C demarcating the maximum wedge range (see Section 5.5 for an explanation for how this range is chosen). Using this technique, it is no longer necessary to add the LTG node T_g along the vector toward the goal, since a subpath directly towards the goal (in the case of no intervening obstacles) can be generated by simply truncating the straight line path between x and T.

5.4.3 Plight of the Nav Cams

Unfortunately, the advantages of using the mast cameras for longer range planning are not as apparent in the realm of the nav cameras. Although the nav cameras can "see" beyond some obstacles, their lower vantage point prevents the generation of as rich an obstacle list. This shortcoming is ameliorated when the nav view is combined with a mast view, as in our RoverBug wedge definition (which also combats the mast cameras' inability to detect obstacles close to the rover).

However, another significant disadvantage of the nav cameras is that they are unable to be panned, without turning the rover itself. Use of the mast cameras is

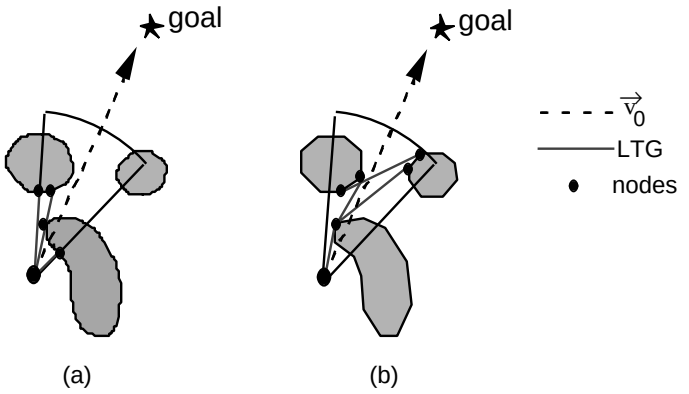


Figure 5.7: Illustration of the differences in nodes used for motion planning between Wedgebug (a) and RoverBug (b). In Wedgebug, the obstacles block both motion and sensing. RoverBug, on the other hand, allows for obstacles which block motion, but **not** sensing. Note that RoverBug also does not consider non-tangent nodes on the boundary of the visible wedge.

not appropriate for those situations in which the rover is expected to skirt obstacle boundaries, which requires awareness of the rover's proximity to close obstacles. In Wedgebug, these situations comprise the "normal" *BF* and "sliding" *MtG* subsegments. In the case of RoverBug, the need for circumnavigating an obstacle in such a manner arises when forward progress toward the goal is impeded by an obstacle which spans the width of the visible region. We have designed an attempt to achieve the objectives of the "normal" *BF* mode using generally only the nav cameras, though more work is needed to reduce the deleterious effect upon allowable obstacle density for this technique to succeed.

5.4.4 The Rover Is Not A Point

Rounding out the high-level discussion of the differences between RoverBug and Wedgebug is a description of how obstacles are handled, in light of the fact that 1) the rover is not a point robot, and 2) the rover can sense much of the extent of the visible obstacles, rather than simply the range to the closest edge. The obstacles are initially detected, using a step/slope model, and segmented into distinct obstacle "blobs" (collections of coordinate points belonging to a single detected obstacle) by a stereo vision algorithm developed by Larry Matthies and Todd Litwin at JPL [48].

Next, we calculate the convex hull of each "blob" in the ground plane (assumed in this implementation to be the XY plane). Finally, we construct the "silhouette" of each convex hull: the projection of the three-dimensional C-space obstacle corresponding to the convex hull and the real rover (see Chapter 3) onto the ground plane. (See Figs. 5.8, 5.9.) (This technique is well-known, often referred to as "growing" the obstacle boundaries.) We have included an empirical safety buffer in the "growth" stage to account for both sensor uncertainty and slight slippage during path execution, as well.

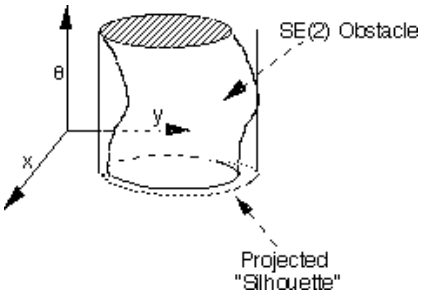


Figure 5.8: Silhouette of a C-space obstacle.

Once the obstacles have been "grown," the algorithm checks whether the goal lies within one of the "grown" convex hulls. If so, the entire convex hull is marked as the goal, allowing the "rover driver" to designate a particular rock as a target, say for instrument placement or analysis. Next, the algorithm determines whether each convex hull lies entirely within the visible wedge; vertices outside the useable region are tagged accordingly. Finally, overlapping obstacles are "melded" into non-overlapping, not necessarily convex "super obstacles," and this reduced obstacle list is checked for wedge-spanning blocking obstacles: that is, obstacles which contain vertices lying outside the bounding edges of the wedge on both sides.*

Finally, we note that the Wedgebug algorithm's decision whether to use "virtual"

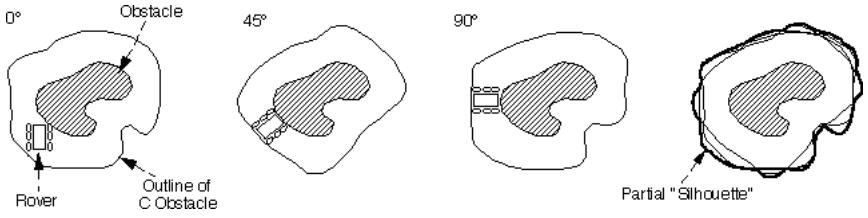


Figure 5.9: Illustration of building a C-space obstacle/silhouette.

*The step of "melding" overlapping obstacles may be discarded in future versions of RoverBug, in favour of other techniques for detecting wedge-spanning obstacles. Although the current approach has the advantage of reducing the total number of obstacle vertices, the "melding" algorithm is complex and lengthy, and is therefore a candidate for code reduction.

MtG or "virtual" *BF*, when faced with a blocking obstacle, essentially relies upon the measurement of the tangents to the obstacle at the edges of the visible region—arguably an imprecise measurement in the real world. Thus, we chose in RoverBug to combine the two "virtual" components into a single "virtual sliding" (*VS*) mode which serves as the transition between *MtG* and *BF*. This change has the added benefit of reducing the overall footprint of the code, a valuable result considering the scarcity of on-board storage.

5.5 Implementation—the Gory Details

In brief, the scenario is as follows: The rover is situated in unknown, rough terrain. The remote human operator designates a goal, using, for example, panoramic rover imagery returned earlier, descent imagery from the lander, or imagery from an orbiter. This action sets in motion the autonomous planner. The planner—initially in *MtG* mode—begins by directing the mast to image towards the goal. Software on-board produces a rangemap from the stereo imagery, and detects "obstacle points"—pixels in the range image determined to be part of an obstacle—using a simple height/slope model within the range image. Additional functions segment the detected obstacle points into discrete 2D connected obstacles, and computes the obstacles' convex hulls. The planner then computes the obstacles' silhouettes, searches the resulting LTG using an A* algorithm [63], and senses additional wedges as needed to produce the first subpath. Before and after the execution of each subpath, the localisation procedure is invoked, to verify the rover's new position. Then, the rover is directed to image toward the goal, and the process repeats, incrementally building subpaths until the goal is reached. (See Fig. 5.10 for a high-level overview of RoverBug's operation.) "RoverBug" is amenable to varying levels of autonomy, from single-step paths under tight operator control, to full multi-step autonomy as described here.

The RoverBug implementation essentially consists of three major pieces of C code which reside within ControlShell—Obstacle Map Pre-Processing, Path Finding,

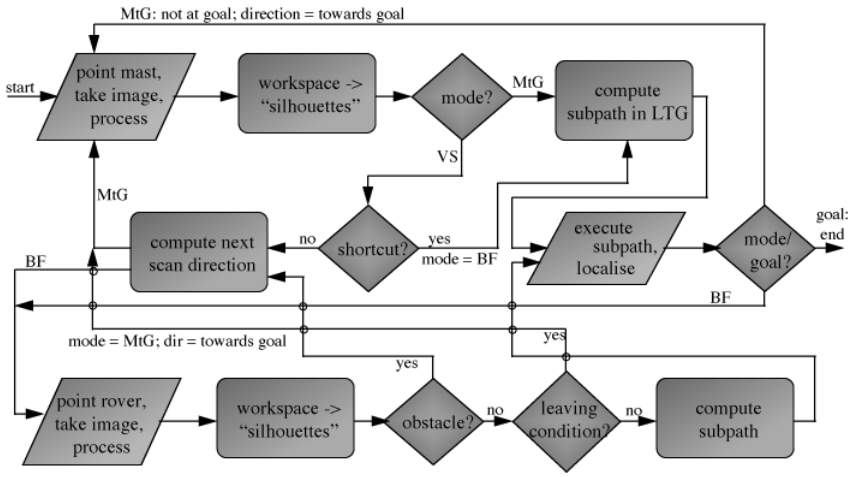


Figure 5.10: High-level flowchart of RoverBug code. Basically, the top row represents the motion-to-goal (MtG) mode, the bottom row shows the operation of boundary following (BF), and the loop in the upper left depicts "virtual sliding" (VS).

and Boundary Following—tied together by an operations loop at the executive level (within SCE). The remainder of this section will describe each piece of the RoverBug code in turn.

5.5.1 Obstacle Map Pre-Processing

After the executive directs the rover to sense a new visual wedge, stereo triangulation is used (as described in Section 5.2) to derive an elevation map of the visible area. A step/slope obstacle model [45], [48] is applied to the range image, to identify those pixels which depict a positive obstacle; these pixels are marked in a bytemap representation. (In the case of a wedge composed from a set of distinct views, at this point the maps for each view are combined into a single bitmap (in the ground, rather than the image, plane) for the wedge.) This representation is then used to segment the obstacles into distinct "blobs," denoting separate obstacles. Next, the X-Y-Z coordinates of the pixels marking the base and top of each obstacle are entered into an array of obstacle points, accompanied by an array of labels indicating which obstacle is associated with each point. (The base and top pixels are represented in the bytemap by different symbols than the bulk of the obstacle, so as to reduce the

size of the obstacle point array while still yielding information about the rearward extent of the obstacle.) Finally, the convex hull of each obstacle's set of points in the array is computed, creating a minimal representation of the detected obstacle. The convex hull points (with their obstacle labels) are entered in an array, which serves as an input to the RoverBug code.

The convex hull array is processed by a RoverBug subroutine, "Work2Cspace," (corresponding to the block labelled "workspace → silhouettes" in Fig. 5.10). An overview of the actions taken during this subroutine appears in Table 5.2. "Work-2Cspace" first creates a pseudo-C-space respresentation of the obstacles by "growing" the convex hulls to approximate their silhouettes (and to include an empirical safety buffer). The "growing" process may create an excessive number of vertices for a given obstacle, a problem both due to the fact that an increase in the number of vertices will increase the time required to search for a path through the obstacles, and since the rover is unable to accurately execute a series of extremely short traverses. Therefore, a minimum distance between "grown" obstacle vertices is enforced. Next, the algorithm determines whether the goal coordinate lies within one of the "grown" convex hulls (denoted "GChulls"); if so, every vertex of that GChull is labelled as a goal. This action allows the rover operator to select a particular rock,

Work2Cspace

1. "Grow" convex hulls to approximate silhouettes (GChulls)
2. Filter GChull vertices to enforce minimum distance requirement
3. Mark vertices of GChull which contains goal
4. "Meld" GChulls which overlap into composite obstacles
5. Mark concave vertices, and those outside visible region ("hidden")
6. Check for composite obstacle with "hidden" vertices on both sides of wedge
7. **If** ∃ blocking obstacle, then
   (a) mode → VS
   (b) compute next direction to be sensed
8. **Else**, **If** mode = VS, then mode → $VS2$

Table 5.2: Overview of the Work2Cspace subroutine

say, as a target, perhaps for later analysis. At this point, the algorithm determines whether any of the GChulls overlap. In the early version of the RoverBug implementation, "Work2Cspace" simply labelled those vertices which lay in the interior of other GChulls; the current version actually "melds" the overlapping GChulls into (not necessarily convex) "super obstacle" polygons. (As previously mentioned, after implementing this feature, it was found that the "melding" code is complex and lengthy; therefore, a future version may bypass this step and return to simply labelling "interior" vertices.) Finally, those polygon vertices which are concave and/or which lie outside of the wedge boundaries (determined currently by "melding" the individual quadrilaterals which demarcate each view) are labelled appropriately.

For the next step in the current implementation, "Work2Cspace" checks the final polygon list for a "super obstacle" which spans the entire visible region (a *blocking obstacle*). (This action corresponds to the two decision blocks, marked "mode?" and "shortcut?" in Fig. 5.10) If there is no such obstacle, and the behaviour mode is "motion-to-goal," control passes to the "Path2Goal" subroutine. If the mode is "virtual sliding," then an "edge" of the previously detected blocking obstacle has been found. The behaviour is changed to "VS2," the closest distance to the goal along the (near side of the) visible obstacle boundary is computed (d_{reach}), and "Path2Goal" is used to find an initial path around the blocking obstacle.

If there is a blocking obstacle spanning the entire wedge, then if the behaviour mode is "motion-to-goal," it is changed to "virtual sliding" (and if it is already "virtual sliding," it remains so). The next direction to be sensed by the rover is computed (corresponding to the "compute direction" block in Fig. 5.10), and control is returned to the executive level to carry out this command. (The response when the behavioural mode is "boundary following" will be discussed in Section 5.5.3.)

The inputs to this subroutine include: the "workspace" convex hull vertex list, the number of workspace obstacles, the start and goal coordinates, the number of distinct views and their FOV polygons, and the current behavioural mode. The subroutine also uses the following parameters: the "growth" radius, the minimum distance between vertices (on a single obstacle), the wedge half-angle, and the em-

parameter	default value
buffer radius	0.4952m
vertex d_{min}	0.5m
emp. range limit	5m
α_{mast}	21°
α_{nav}	40°

Table 5.3: Default values for RoverBug parameters

pirical range limit for the wedge[†]. (See Table 5.3 for the default values of these parameters used for Rocky7.)

The outputs are the final obstacle polygon list, the number of polygons, and the array of polygon vertex labels; d_{reach}; the index of the next direction to be scanned as well as the "positive" sense of rotation; and the next behavioural mode.

5.5.2 Finding a Path

Once it is determined that no obstacle spans the entire wedge, a RoverBug subroutine called "Path2Goal" (corresponding to the "compute subpath" block in Fig. 5.10) is used to determine the locally optimal path to the goal—that is, the shortest length path, considering only the visible obstacles. (See Table 5.4 for an overview.) This subroutine implements an A* graph search, which is summarised as follows: Two linked lists are maintained: a *Closed* list tracks the nodes of the LTG which have been expanded, and an *Open* list contains those nodes which have been "visited," but have not been expanded. Expansion of a given node V consists of finding those nodes Y which neighbor V on the LTG; computing the "backward cost" of a path from the current rover position x to each Y and the estimated "total cost" of a path from x to T through Y; and finally entering Y with its associated costs (and with V as the "previous" node) into the Open list. The Open list is sorted by total cost, and the node with the least total cost is moved to the Closed list and expanded. The process is initialised by entering x into the Open list, and ends when a goal node is entered in the Closed list. In this manner, the subroutine executes a

[†]This number acts as a limit on the length of each subpath, and may be used when concerns other than the effective range of "good" stereo data arise, as seen in Section 5.6.

Path2Goal

1. Initialise open and closed lists
2. **Do** (until open list is empty or V = goal):
   (a) Copy top of open list (node V) to bottom of closed list
   (b) "Expand" V:
      i. Find convex vertices Y adjacent to V in LTG
      ii. Compute backward and total costs for each Y
      iii. Add each Y to open list
   (c) Sort open list by total cost
3. Construct path from closed list
4. **If** $\exists$ path to goal, then
   (a) **if** mode = MtG, truncate path at maximum visible range
   (b) **if** mode = $VS2$, mode $\rightarrow$ BF
5. **Else**, return ERROR

Table 5.4: Overview of the Path2Goal subroutine

depth-first search for the shortest path to the goal through the LTG. The final path is found by reading out backwards from the Closed list, beginning with the goal node; this path is truncated at the maximum allowable range to create the subpath.

In the early implementation of RoverBug, if the path search ends by exhausting the LTG nodes in the Open list before finding a path to a goal node, the planner switches to an ad-hoc boundary following behaviour, described in the next section. In the current version, such a situation would herald an error condition (since being unable to find any clear paths indicates the existence of a blocking obstacle, which should have been found by "Work2Cspace"), and the algorithm would halt. Otherwise, if the behaviour mode is "VS2," it is switched to "boundary following."

The inputs to "Path2Goal" include: the obstacle polygon vertex list and labels, the number of polygons, the start and goal coordinates, the global parameter d_{LEAVE}, the empirical range limit, and the current behaviour mode.

The output consists of the list of waypoints constituting the subpath, the number of waypoints, and the next behaviour mode.

5.5.3 Boundary Following

In the early implementation of RoverBug, which was meant as a validation of the LTG approach, not much consideration was given to the boundary following mode, particularly since it would necessarily rely upon the low-mounted, fixed nav cameras. Therefore, when faced with a blocking obstacle, the rover was directed to advance to the end of the subpath which brought the rover closest to the goal, and then simply to move 2m left or right (relative to the direction towards the goal from x), based upon in which half of the wedge the last waypoint lay. While the subpath could be executed using direct rover motion commands, the 2m sidestep always used the "Go-to-Waypoint" algorithm in order to avoid collisions. The rationale was that, given the types of terrain which we expected the rover to encounter, it was possible that a blocking obstacle could be avoided by simply gaining a different, nearby viewpoint.

However, in order to bring RoverBug back into alignment with the Wedgebug theory, and to restore a notion of convergence, the current implementation includes a boundary following subroutine (depicted by the bottom line of Fig. 5.10, and sketched in Table 5.5). The underlying structure echoes the "virtual sliding" loop described above, replacing swiveling the mast with turning the rover. Although using rover motion to obtain additional views is not ideal for many reasons, at the very least the vehicle's center of rotation is near the forward-driving end, so the collection of nav camera views nearly form a circle with a consistent maximum range.

At the start of each boundary following step, the rover turns to take a nav image towards the goal. (It should be noted that by design, the nav view should contain part of the previously sensed obstacle boundary, to ensure that no potential passage is missed.) After executing "Work2Cspace," the planner has determined whether or not an obstacle spans the visible region. If there is no such obstacle, the mast is deployed, and senses a wedge in the direction of the goal. If again there is no blocking obstacle, then the planner searches for an LTG node V which satisfies the *leaving condition*: that is, if $\mathrm{d}(V,T) < \mathrm{d}_{reach}$, then RoverBug resets $\mathrm{d}_{\mathrm{LEAVE}}$ to

BoundaryFollowing

1. Sense and process nav camera image toward goal
2. **If** executed loop, return GOAL UNREACHABLE
3. **If** $\nexists$ blocking obstacle, then
   (a) **If** rover is facing goal, then
      i. Deploy mast; image towards goal
      ii. **If** $\nexists$ blocking obstacle
         A. **If** $\exists$ node V which satisfies *leaving condition*, then mode $\rightarrow$ MtG; return
      iii. **Else**, mode $\rightarrow$ VS; return
   (b) **Else**, compute farthest rover can progress along boundary; return
4. **Else**, compute direction to turn rover for next nav view
5. **If** new direction will sense previously sensed area, return ERROR

Table 5.5: Overview of the BoundaryFollowing subroutine

$\mathrm{d}(V, T)$, and changes the behaviour mode to "motion-to-goal." Otherwise (in the cases that no such node can be found or there exists a blocking obstacle in the mast view), the behaviour switches back to "virtual sliding," preserving the previously established "positive" sense of rotation and sensing additional views only in this direction. In this manner, the planner may produce a shortcut around the blocking obstacle, taking advantage of the fact that the mast has already been deployed.

If, on the other hand, there is a blocking obstacle in the nav view, the rover is turned by twice the nav wedge half angle, α_{nav}, and a new view is sensed and combined with the prior wedge. Again, if there is still a blocking obstacle, the rover repeats this action, turning and sensing a new view. If the rover has turned so far that it is resensing a region already included in the combined wedge (tracked using the sunsensor), and still cannot find a clear path, the algorithm halts and signals an error condition. In addition, if at any time during boundary following the rover senses the point V_{loop} and detects that it has circumnavigated the obstacle (using sunsensor data to track a "winding number"-like parameter), the RoverBug algorithm halts and signals that the goal is unreachable. The winding parameter records the difference between the rover's current heading and its "original heading,"

represented by the vector between V_{loop} and the goal of the $VS2$ subpath, as the rover traverses around the obstacle. The object is to ensure that the rover will take action only if V_{loop} is encountered in the correct orientation; that is, the algorithm will not halt prematurely if the (concave) obstacle boundary curves back towards V_{loop}.

Finally, in the case that the rover has turned and detects no blocking obstacle, the planner computes the farthest the rover can advance along the (former) blocking obstacle boundary, and updates d_{reach}. The rover executes this subpath, and begins a new boundary following step.

The inputs to "BoundaryFollowing" include: the obstacle polygon list, labels, and number of polygons, the start and goal coordinates, d_{reach}, and the current behaviour mode.

The outputs include the list (and number) of waypoints for the subpath, and/or the next direction to be scanned by the nav or mast cameras, and the next behavioural mode.

5.5.4 Tying It All Together

The executive level within SCE manages the high-level loop which drives the Rover-Bug planner until the goal is reached or the algorithm otherwise halts. This level, in essence, acts as a "switch" statement, invoking actions based upon the current behaviour mode of the planner. To wit: if the planner has returned a subpath, SCE directs the rover to execute the list of waypoints, using either direct rover motion commands or the "Go-to-Waypoint" algorithm. (The "Go-to-Waypoint" algorithm is generally used for the first waypoint, if the wedge towards the goal does not include a nav view.) At the end of the subpath, if the behaviour mode is "motion-to-goal," the rover executes the second step of the localisation algorithm (imaging back toward x) and updates its position estimate, then images toward the goal for the first step of the next localisation command. Next, if the behaviour mode is

- *motion-to-goal* or *virtual sliding*, then the executive deploys that mast and senses a wedge in the direction indicated by the given vector index k and the

positive sense of rotation. That is, the mast senses along the vector $\vec{v}_k$, where $\angle(\overrightarrow{xT}, \vec{v}_k) = 2k\alpha,\ k \in \mathbb{Z}$. The images are processed using the stereo triangulation and obstacle detection code described earlier, and if the behaviour is "virtual sliding," the resulting obstacle list is combined with the prior data to create a conglomerate wedge.

- *boundary following*, a secondary cue indicates whether the executive should turn the rover and take an image with the nav cameras (in a similar manner as above), or should deploy the mast and image towards the goal.

In either case, after taking the appropriate action, the executive again calls the RoverBug function, and awaits its next directive.

5.6 Results, or Rocky Goes For a Sunday Drive

The early implementation of RoverBug has been tested extensively in the JPL MarsYard, as well as in natural arroyo terrain, validating the "motion-to-goal" portion of the RoverBug planner. The wedge was defined as a single mast view, with the tilt angle chosen according to the downrange footprint of a mast camera FOV. We chose a tilt angle of 25°, which yields a theoretical footprint from 1.6m to 8.7m. (The actual coverage was somewhat larger in both directions, with the densest range data from roughly 1.5m up to 6m.) A range limit R for each wedge was calculated using the minimum of a) the theoretical maximal range for the FOV, and b) the range at which an obstacle 18cm tall subtends 10 pixels; for the current mast setting, the range limit was 7.8m. Later testing indicated that localisation was most effective for traverses within 5m, so this distance was added as an empirical range limit. The mast camera tilt angle was adjusted accordingly, to 29°.

Figure 5.11 shows the results of one typical multi-step run in the MarsYard. The MarsYard is an outdoor area, roughly 20m square, which features a sandy substrate and volcanic rocks, simulating martian terrain. The rocks are arranged according to a computer-generated pattern, meant to simulate the sizes and placements of

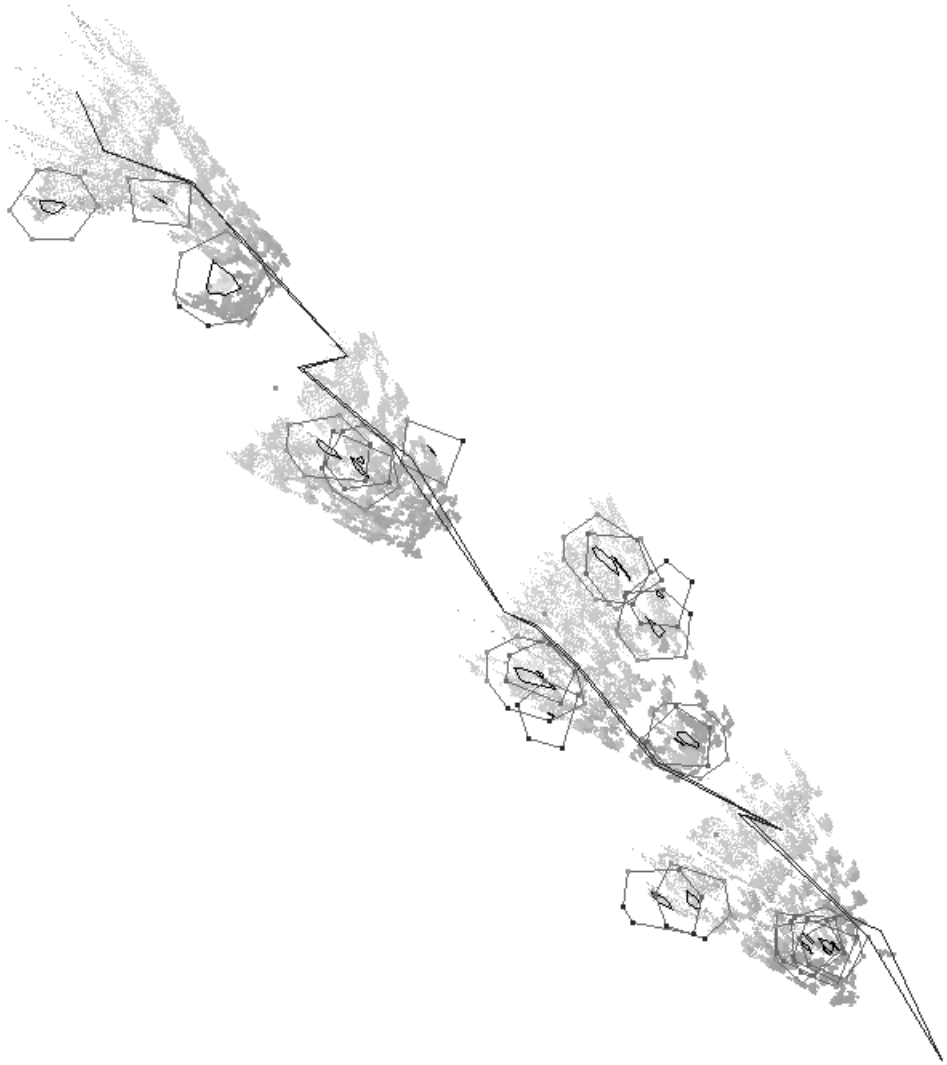


Figure 5.11: Results from a multi-step run in the JPL MarsYard. The path begins in the lower right corner of the image, toward a goal approx. 21m distant in the upper left. Each wedge depicts a rangemap produced from mast imagery, and extends roughly 5m from the imaging position. The obstacles are marked by a black convex hull, and a grey silhouette. Each subpath ends with an apparent "jag" in the path; these are not in fact motions, but rather the result of the localisation procedure run at the conclusion of each step. The second line echoing the path is the rover's telemetry for the run.

rocks for the type of terrain encountered by Viking Lander-1 (VL1).[‡] The goal was approximately 21m distant from the initial position, and R for each wedge (the radius of useable data) was 5m. The rover's initial position was in the lower right corner. As in Fig. 5.5, the convex hulls and silhouettes were computed within each wedge view, and a subpath generated as described above. This subpath was executed before the rover's mast was redeployed for localisation. The results of the localisation procedure appear in the figure as "jags" in the path at the end of each subpath. These features are not actual motions, but rather updated final positions. The rover was next directed by the planner to image toward the goal from his new position. This process was repeated, for a total of four wedges and subpaths, until the goal was achieved. The resultant multi-step path runs from lower right to upper left. The time taken by this entire traverse—from the initial sending of the autonomous navigation command (with a 1m tolerance in achieving the goal) to the completion of the final localisation step—was 57 min. and 47 sec. On average, the planning of each subpath (from the time the planning command was sent to the time telemetry was received) took 3.25 sec. For comparison, the first localisation procedure (including both pre-traverse and post-traverse steps) took 6 min. Vision processing, as well as the slow traverse speed of the rover, contributed most of the time spent during this run.

Note that the first 1–1.5m at each rover position was not sensed, due to the choice of tilt angle, which in turn was based upon the limitations of the mast-based obstacle detection process. In the particular run shown in Fig. 5.11, direct motion commands (e.g., drive forward x meters) were sent to the rover to execute each subpath.

However, in later runs, e.g., the long traverse shown in Fig. 5.12, the "Go-to-Waypoint" algorithm was used as an ad-hoc collision avoidance mechanism when the rover drove from its current position to the first subpath waypoint. This traverse, which took place in the arroyo near JPL, was toward a goal 44m distant. As can be

[‡]At present, only the largest rocks have been placed; in the future, smaller rocks will be used to segment the MarsYard into regions simulating VL1, VL2, and Pathfinder-like terrains.

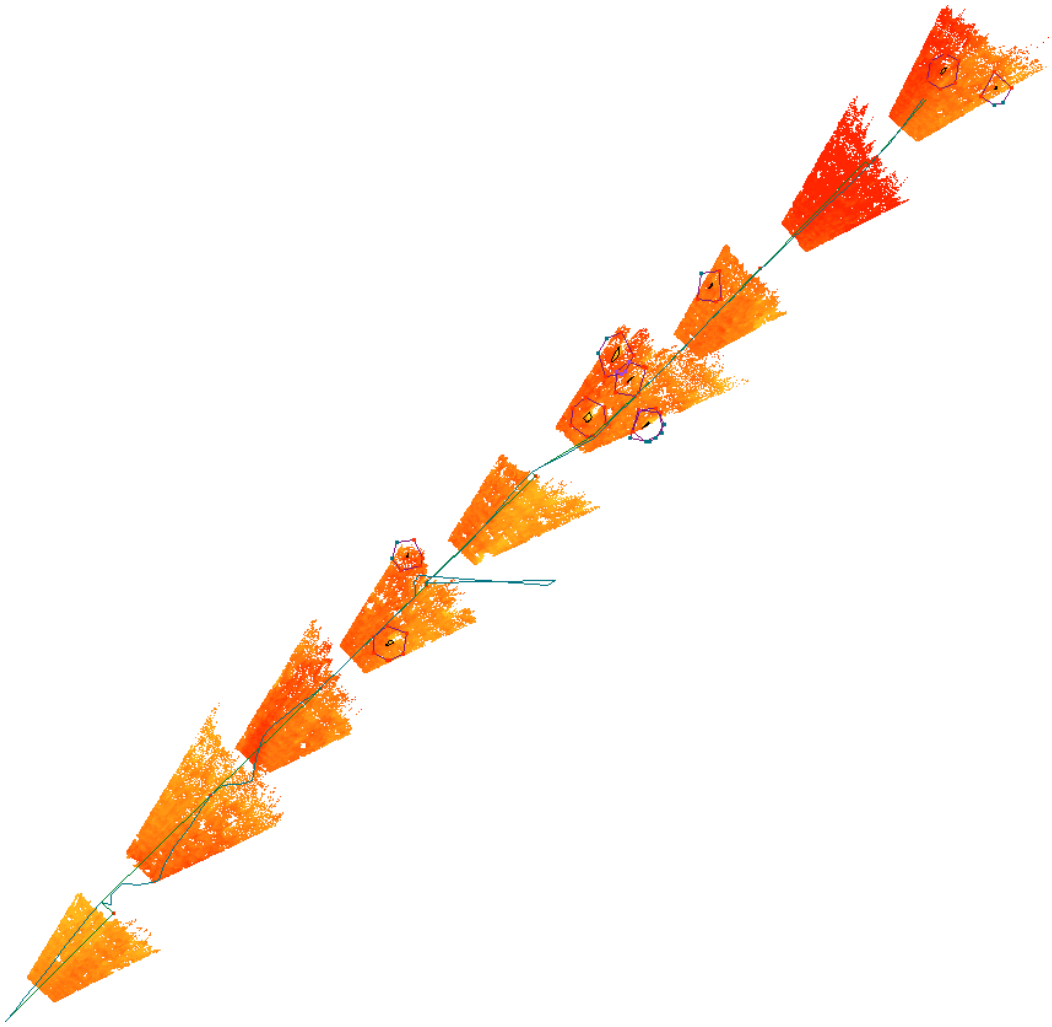


Figure 5.12: Results from a multi-step run in the arroyo near JPL. The path begins in the lower left corner of the image, toward a goal approx. 44m distant in the upper right. Each wedge depicts a rangemap produced from mast imagery, and extends roughly 5m from the imaging position. The obstacles are marked by their convex hulls and silhouettes. Again, the localisation procedure updates the position estimate after most steps. In this run, however, the initial waypoint of each subpath is executed using the "Go-to-Waypoint" command, to avoid collisions with unsensed obstacles in the first 1.5m of each wedge. After the fourth wedge, this command caused the rover to become "stuck" near a rock, causing the apparent lengthy detour from the prescribed path. The second line echoing the path is the rover's telemetry for the run.

Figure 5.13: A typical view from the nav cameras during the 9-step run

seen in the figure, the use of the "Go-to-Waypoint" command resulted in one case (after the fourth step) in the rover becoming "stuck" near a rock for some time, resulting in a long apparent detour away from the intended path (as the drift in the rover's position estimate increased). The heuristic employed by the collision avoidance mechanism caused the rover to repeatedly turn away from, then towards the same rock in its efforts to reach the first subpath waypoint. In fact, the run was temporarily aborted at this point, then continued with the rover in a slightly different position (and with its position estimate reset). Otherwise, as can be seen, the extended traverse proceeded smoothly. Fig. 5.13 shows a typical view from a nav camera during the run. It should be stressed that the use of the "Go-to-Waypoint" algorithm is in response to the need for collision avoidance during these experiments, and is not part of the RoverBug implementation. A better path-following algorithm (the "piloting" level described in Section 5.3) is needed in order to allow collision avoidance without adversely affecting the convergence properties of the RoverBug algorithm.[§]

Thus, the general approach of the RoverBug algorithm has been validated through experimentation, and yields good performance in navigation to distant goals.

To give a very rough estimate of the memory requirements of the RoverBug algorithm, currently, the Work2Cspace code uses 82 KB of static memory, and Path2Goal uses 89.6 KB. In addition, Path2Goal utilises roughly 5.1 KB of dynamically allocated memory. However, no effort had been made to optimise the memory

[§]Indeed, the performance of any high-level path planner would be improved by a better piloting algorithm.

usage of this experimental code, and straightforward modifications can significantly reduce the memory load. For example, if the code is modified to dynamically allocate the currently static memory, then for a typical case (10 obstacles per wedge, with 25 vertices on each obstacle, resulting in a path with 10 waypoints) the memory usage drops to a total of 8.5 KB for Work2Cspace, and 15.1 KB for Path2Goal.

5.7 Summary

This chapter describes the Rocky7 testbed vehicle, which provides a strong analogue to flight planetary microrovers. In particular, the sensing suite, consisting primarily of stereo pairs of cameras mounted at two heights with differing fields of view, accurately captures the sensing scenario expected for the Mars rover missions currently being planned. Due to the idiosyncracies of this sensing paradigm, as well as practical considerations, the Wedgebug algorithm described in Chapter 4 must be modified in order to create a practical implementation. We discuss the specific changes necessary, and describe the implementation, dubbed RoverBug, in some detail. Finally, we present results from traverses in the MarsYard, an outdoor martian terrain simulant, and in natural (arroyo) terrain. These results validate the LTG approach using actual data, and show good performance in martian-like terrains.

Chapter 6

And Now We Have Arrived, But Have Only Just Begun

path·find·er (păth′fīn′dr) *n. One who discovers a way through or into unexplored regions.*
—The American Heritage Dictionary

6.1 Conclusion

Although the autonomous robots dreamed of by science fiction may yet be far off into the future, the time is coming soon when robots will explore distant worlds without constant human oversight. These robots must be able to manoeuver in rough, unknown terrains and to reach their designated targets safely and successfully, using autonomous sensor-based motion planning techniques. The Sojourner rover—despite her simplicity—represented the most autonomous spacecraft launched up to that time (December 1996), with her ability to determine her own route based on sensor information. Her pathfinding mission embodied the first tiny cleat marks* toward the full realisation of robotic planetary exploration.

Future Mars explorers will continue to operate under severe constraints compared to their Earthbound counterparts, so the key to mission success will be algorithms which adhere to these constraints yet produce highly capable machines able to traverse extended distances through rough, unknown territories. We have

*or, in human terms, baby steps

developed the Wedgebug algorithm, a sensor-based motion planner tuned to the constraints of flightlike microrovers, such as limited computational resources and scarce memory, yet provably complete and correct. This planner also has the advantage that it is able to effect locally optimal (shortest length) paths, considering only the visible obstacles. Wedgebug is appropriate for point robots, equipped with range-finding sensors having limited downrange and angular scope, inhabiting unbounded planar environments, where the region surrounding the goal (including the robot's initial position) is populated by a finite number of obstacles, each with piecewise C^1 boundaries. Wedgebug utilises a streamlined, temporary local world model—consisting primarily of sensed obstacle boundary endpoints—and a small set of global parameters to ensure convergence to the target while minimising the algorithm's memory requirements. In addition, Wedgebug incorporates automatic gaze control, sensing only those regions required for efficiency, which provides the dual benefits of minimising computationally- and memory-expensive sensing while conserving motion, which is costly due to energy use, drift, slippage, and other localisation errors.

We have also developed an extension to Wedgebug, denoted RoverBug, which is directly relevant to the Mars missions currently being planned. The RoverBug planner is the implementation of the Wedgebug concept on an actual prototype planetary microrover, the Rocky7 vehicle developed at the Jet Propulsion Laboratory precisely for such pursuits. While retaining many of the key properties of Wedgebug: being correct, complete (with respect to the obstacle "silhouettes") and producing locally optimal paths while minimising computation, sensing, and rover motion; RoverBug also takes into account the idiosyncracies of the microrover-class sensor suite, including the ability to "see over" many obstacles and the lack of panable proximity sensors. The approach has been verified by experiments in natural terrain, including test sites both in an arroyo near JPL and in the MarsYard, an outdoor martian terrain analogue. As a consequence of these developments, we have delineated many of the issues and constraints faced by motion planners for planetary microrovers, which we hope will serve as a useful guide for additional work in this

area.

Finally, we have contributed a corrected, more detailed proof of a key result[†] required for the proof of completeness for both Wedgebug and the prior Tangent Bug algorithm developed by Kamon, Rimon, and Rivlin [21], and have extended the domain of application to a broader set of obstacles and environments. Several sensor-based motion planners, which depend upon the use of distinct modes to escape dead-ends while traversing towards a goal, rely upon the claim that there are a finite number of such dead-ends in order to prove completeness. We have proven this conjecture for the case of a planar environment and obstacles with piecewise C^1 boundaries in not necessarily generic arrangements, with a range-bounding condition imposed upon the robot by the planner. (This last requirement is not onerous, since such a condition is used to track and maintain progress toward the goal.)

6.2 Future Work

Although the work presented in this thesis demonstrates an improvement in the state of the art of motion planners for flightlike planetary microrovers, Wedgebug and RoverBug are still just the first steps toward the goal of highly capable planners for this class of vehicles, able to handle extensive traverses through varying types of planetary terrain. We describe here several useful directions for future research.

6.2.1 Boundary Following

In the short term, we plan to extend our experimental validation of the RoverBug approach by incorporating the virtual sliding (VS) and boundary following (BF) modes described in Chapter 5, whose validation has been prevented earlier by practical constraints. The BF mode's algorithm as given is a first attempt to compensate for the lack of side-viewing proximity sensors on Rocky7, as well as limitations in sensing regions close to the rover using the mast, while trying to minimise the number of short traverses (costly in terms of causing dead-reckoning drift) and of

[†]originally presented in [21] with an informal proof

mast deployments (the mast cannot be in the raised position while the vehicle is in motion, and gross mast movements are slow). The next step is to refine this algorithm, perhaps using stand-off distances from the blocking obstacle and alternate orientations to allow the rover to view more of the obstacle without necessitating large turns-in-place. Such refinements, besides potentially reducing the number of expensive short steps around an obstacle, could also allow the rover to operate in more cluttered environments than would be possible with the current approach.

6.2.2 Extension of Wedgebug to SE(2)

Wedgebug (and RoverBug) models planetary terrain as a planar configuration space, segmented into traversable terrain, or freespace, and impassable regions, marked as obstacles. While this representation is efficient, it does not capture other difficulties presented by the environment, such as terrain features which pose varying risks to traversal depending upon the rover's orientation. Therefore, it would be desirable to extend these algorithms to $SE(2)$[‡]. RoverBug does include a first step toward such an extension, by computing the "silhouettes" of the obstacles. The resulting algorithm is correct, but not entirely complete; the silhouettes may mask a passage to the goal which could be achieved by assuming the proper rover orientation. A short term goal would be to analyse the extension of Wedgebug to silhouettes; a longer term research direction is the extension of the algorithm to full $SE(2)$. Among the implementation issues for an $SE(2)$ RoverBug is the problem of computing an $SE(2)$-obstacle from the vehicle's sensor data, which typically returns only the face of the obstacle in the sensor's direction.

6.2.3 Traversability

Besides the limitation described above in the algorithms' world model, this simple model also does not allow optimisation of, say, the energy required to traverse a particular patch, nor does it allow the classification of the environment into varying risk levels for traversability. A clear evolutionary step would be the definition of

[‡]i.e., the space of 2D translations and rotations

several terrain types, assigned appropriate risk levels, from analysis of visual data using cues such as texture or slope variation. Another issue is the fact that the planners developed here do not account for sensor uncertainty; an aspect which could perhaps be included in the assignation of risk level to terrain types. Two approaches for using the current algorithm with these new terrain classes are as follows:

- Iterate the planner, first designating an initial risk level above which terrain classes would be treated as obstacles; if no clear path can be found, raise the risk level gradually and attempt a replan until an upper risk threshold has been reached, in which case boundary following would be initiated (or continued).
- Use the planner to generate an initial seed path, to be optimised over such variables as time, energy, or risk.

Future refinements could weave consideration of traversability issues more tightly into the fabric of the planner itself.

6.2.4 Localisation

Mobile robots are often plagued by errors in dead reckoning, the process by which a robot uses its measurements of odometry and heading, as well as any estimates of drift, to determine its location relative to an absolute coordinate system. This problem is exacerbated on planetary missions (compared to earthly applications) by the absence of GPS-like systems.[§] Although the Wedgebug algorithm somewhat ameliorates the effect of this drift by not requiring newly aquired data to be registered against a global world model, the method does bookkeep the goal position as a coordinate point, and tracks another coordinate to determine whether the robot has executed a loop around an obstacle. Work has been done on autonomous localisation techniques, including Olson's localisation algorithm [59] and the kinematic

[§]There are preliminary plans in the works, however, to send a collection of GPS-like satellites in orbit around Mars, perhaps as soon as 2001.

state estimator developed by Balaram [3], both implemented on Rocky7. However, further improvements in rover localisation are desirable. Similarly, issues of target recognition arise when the rover is commanded, say, to return to the lander or to some other previous location. In these situations, visual servoing could be used to augment localisation.

A related issue is the designation of a target as something other than a coordinate point—for example a terrain feature, a desired heading (with or without an associated distance), or even an item of scientific interest determined by sensor or instrument readings—and how to treat these alternate designations as goals within the Wedgebug framework.

6.2.5 Fine Motion Planning

Once RoverBug has moved the rover to the designated goal location (within an error radius, due to dead reckoning drift), the vehicle may need to adjust its configuration in order to place an instrument at a particular orientation on a rock face, say, or to image a desired feature, or to collect a designated rock or soil sample. At this range, the non-holonomic constraints of the vehicle—ignorable for the most part during longer traverses, since the rover is capable of turning nearly in place and is travelling in primarily one general direction at slow speeds—come into play. A planner specifically geared to this type of motion ("fine motion," or "terminal navigation") is required. Some work in this direction has been done with the Rocky7 rover, particularly in the area of "hand-eye" coordination (coordinating the robot's 2 DOF manipulator with visual servoing) [10], but more work could be done which includes, for example, awareness of nearby obstacles.

A related problem is the design of a piloting level, which can accurately execute the paths produced by the motion planner while avoiding hazards.

6.2.6 On-board Resource Management

Finally, although the Wedgebug and RoverBug planners attempt to mitigate the scarcity of available resources—in their case, computational prowess and limited

memory, as well as considerations of time and minimising dead-reckoning error—much more could be done to allow a planetary rover to determine (and change) activity plans based upon such resources as available power, communication opportunities, and maintaining internal temperature within an acceptable range. Work in this area is underway, using the ASPEN system implemented on the Rocky7 vehicle [2] (among other spacecraft); future work will integrate the RoverBug planner with this scheduler, further increasing the autonomy of planetary surface explorers.

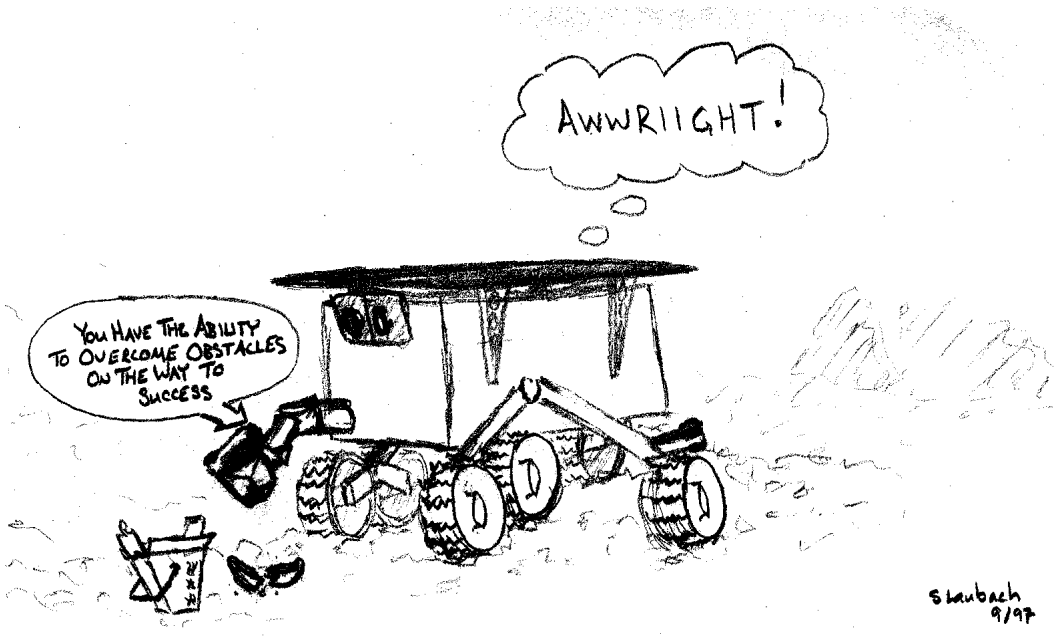


Figure 6.1: Cartoon of Rocky7 drawn by the author, based upon a fortune cookie received during the course of work on this thesis.

Bibliography

[1] AO 97-OSS-04: Mars 2001 Letter of Solicitation. http://www.hq.nasa.gov/-office/oss/ao/97-oss-04/mars2001.htm#Lander and Rover Missions Overview and Objectives. This page, in part, provides an overview of the proposed mission for the 2001 Mars Surveyor Lander and Rover. Sorina Broce, Curator.

[2] ASPEN web site. http://www-aig.jpl.nasa.gov/public/planning/rover. This page provides an overview of the use of the ASPEN scheduler with the various rover programs at JPL.

[3] J. Balaram. Kinematic state estimation for a Mars rover. To be published in a special issue of *Robotica* on Intelligent Autonomous Vehicles.

[4] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, December 1991.

[5] R. Brooks and A. Flynn. A robust layered control system for a mobile robot. *IEEE Trans. on Robotics and Automation*, 2(1), 1986.

[6] B. Brumitt, R. Coulter, and A. Stentz. Dynamic trajectory planning for a cross-country navigator. The Robotics Institute, Carnegie Mellon University, 1992.

[7] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[8] R. Chatila and S. Lacroix. Adaptive navigation for autonomous mobile robots.

In *Robotics Research: The Seventh International Symposium*, pages 450–458. Springer-Verlag, 1996.

[9] H. Choset. *Sensor-Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. Ph.D. thesis, California Institute of Technology, 1996.

[10] H. Das, X. Bao, Y. Bar-Cohen, R. Bonitz, M. Lindemann, M. Maimone, I. Nesnas, and C. Voorhees. Robot manipulator technologies for planetary exploration. In *Proceedings of the Smart Structures and Integrated System Symposium*, Newport Beach, CA, March 1999.

[11] FIDO WWW site. http://robotics.jpl.nasa.gov/tasks/etrover. This page presents technical information about the FIDO Microrover. Eric Baumgartner, Page Maintainer.

[12] E. Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. Ph.D. thesis, Virginia Polytechnic Institute and State University, July 1991.

[13] D. Gennery. Traversability analysis and path planning for a planetary rover. To be published in *Autonomous Robots*.

[14] M. Golombek, January 1999. Mars Pathfinder Project Scientist, personal communication.

[15] M. Golombek and D. Rapp. Size-frequency distributions of rocks on Mars and Earth analog sites: Implications for future landed missions. *Journal of Geophysical Research*, 102(E2):4117–4129, February 25, 1997.

[16] M. P. Golombek, H. J. Moore, A. F. C. Haldemann, T. J. Parker, and J. T. Schofield. Assessment of Mars Pathfinder landing site predictions. *Journal of Geophysical Research, Special Issue on the Mars Pathfinder Mission*. accepted 9/98.

[17] M. Goresky and R. MacPherson. *Stratified Morse Theory*. Springer-Verlag, Berlin, 1988.

[18] V. Guillemin and A. Pollack. *Differential Topology*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.

[19] M. Henle. *A Combinatorial Introduction to Topology*. Dover Publications, Inc., New York, N.Y., 1994.

[20] M. W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems, and Linear Algebra*. Academic Press, Inc., San Diego, CA, 1974.

[21] I. Kamon, E. Rimon, and E. Rivlin. A new range-sensor based globally convergent navigation algorithm for mobile robots. CIS - Center of Intelligent Systems 9517, Computer Science Dept., 1995.

[22] I. Kamon, E. Rivlin, and E. Rimon. 3DBug: A three-dimensional range-sensor based globally convergent navigation algorithm. CIS - Center of Intelligent Systems 9618, Computer Science Dept., 1996.

[23] I. Kamon, E. Rivlin, and E. Rimon. A new range-sensor based globally convergent navigation algorithm for mobile robots. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 429–435, April 1996.

[24] A. Kelly and A. Stentz. An approach to rough terrain autonomous mobility. In *International Conference on Mobile Planetary Robots*, Santa Monica, CA, January 29–February 1 1997.

[25] A. Kelly and A. Stentz. Rough terrain autonomous mobility–part 2: An active vision, predictive control approach. In *Autonomous Robots 5*, pages 163–198, 1998.

[26] A. J. Kelly. RANGER—an intelligent predictive controller for unmanned ground vehicles. The Robotics Institute, Carnegie Mellon University, 1994.

[27] O. Khatib. *Commande Dynamique dans l'Espace Opérationnel des Robots Manipulateurs en Présence d'Obstacles*. Ph.D. thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, France, 1980. In French.

[28] J. R. Killian, Jr. Making science a vital force in foreign policy. *Science*, 33:25, January 6, 1961.

[29] A. N. Kolmogorov and S. V. Fomin. *Introductory Real Analysis*. Dover Publications, Inc., New York, N.Y., 1970.

[30] C. R. Koppes. *JPL and the American Space Program: A History of the Jet Propulsion Laboratory*. Yale University Press, New Haven, Conn., 1982.

[31] E. Kreyszig. *Differential Geometry*. University of Toronto Press, Toronto, Canada, 1959.

[32] T. Kubota, I. Nakatani, Y. Kuroda, T. Adachi, H. Saito, and T. Iijima. Izu-ohshima field tests for autonomous planetary rover. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 588–593, Victoria, B.C., Canada, October 1998.

[33] T. Kubota, I. Nakatani, and T. Yoshimitsu. Path planning for planetary rover based on traversability probability. In *Proceedings of the International Conference on Advanced Robotics (ICAR'95)*, pages 739–744, 1995.

[34] T. Kubota, I. Nakatani, T. Yoshimitsu, H. Katoh, Y. Kuroda, T. Adachi, T. Iijima, H. Saito, and T. Takano. Autonomous behavior control for lunar or planetary rover. In *International Conference on Mobile Planetary Robots and Rover Roundup*, Santa Monica, CA, January 29–February 1, 1997.

[35] D. Langer, J. K. Rosenblatt, and M. Hebert. A behavior-based system for off-road navigation. *IEEE Trans. on Robotics and Automation*, 10(6):776–783, December 1994.

[36] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[37] S. L. Laubach. Notes from experiments with Rocky7 in the JPL MarsYard, Summer, 1996.

[38] Y. H. Liu and S. Arimoto. Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. *Int'l Journal of Robotic Research*, 11(4):376–382, 1992.

[39] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. on Computers*, C-32(2):108–120, 1983.

[40] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[41] Mars Pathfinder Microrover Telecommunications home page. http://mpfwww.-jpl.nasa.gov/MPF/rovercom/rovcom.html. This page presents technical information about the Mars Pathfinder Microrover. In particular, the Frequently Asked Questions section contains useful data about the rover and her mission. Scot Stride, Author.

[42] Mars Pathfinder mission and Ares Vallis landing site. *Journal of Geophysical Research*, 102(E2), February 25 1997. Special Issue.

[43] A. Martin-Alvarez. Control system for planetary mobile robots. European Space Agency - Internal ESTEC Working Paper no: 1811, December 1994.

[44] J. Matijevic, February 1999. Athena Rover Chief Engineer and former math professor, personal communication.

[45] L. Matthies. Passive stereo range imaging for semi-autonomous land navigation. *Journal of Robotic Systems*, 9(6):787–816, September 1992.

[46] L. Matthies, May 1999. Machine Vision Group Supervisor, personal communication.

[47] L. Matthies, E. Gat, R. Harrison, B. Wilcox, R. Volpe, and T. Litwin. Mars microrover navigation: Performance evaluation and enhancement. *Autonomous Robots Journal*, 2(4):291–312, 1995. Special issue on Autonomous Vehicles for Planetary Exploration.

[48] L. Matthies, A. Kelly, T. Litwin, and G. Tharp. Obstacle detection for unmanned ground vehicles: A progress report. In *Robotics Research: The Seventh International Symposium*, pages 475–486, London, 1996. Springer-Verlag.

[49] L. H. Matthies and P. Grandjean. Stochastic performance modeling and evaluation of obstacle detectability with imaging range sensors. *IEEE Trans. on Robotics and Automation*, 10(6):783–791, December 1994.

[50] MGS investigation description and science requirement document. JPL Document D-12487, February 1995.

[51] J. Milnor. *Morse Theory*. Princeton University Press, Princeton, N.J., 1963.

[52] A. Mishkin, fall 1997. 2001 Rover System Engineer, personal communication.

[53] A. Mishkin, fall 1998. 2001 Rover System Engineer; 2003/2005 Rover System Engineer & Rover Operations Lead, personal communication.

[54] A. Mishkin, J. Morrison, T. Nguyen, H. Stone, B. Cooper, and B. Wilcox. Experiences with operations and autonomy of the Mars Pathfinder microrover. In *Proc. IEEE Aerospace Conf.*, Snowmass at Aspen, Co., March 21–28 1998.

[55] M. Morse and S. S. Cairns. *Critical Point Theory in Global Analysis and Differential Topology*. Academic Press, New York, N.Y., 1969.

[56] I. Nakatani, T. Kubota, T. Adachi, H. Saito, and S. Okamoto. Navigation technique for planetary rover. In *Proceedings of the International Conference on Advanced Robotics (ICAR'95)*, pages 201–205, 1995.

[57] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, Washington, D. C., 1969.

[58] C. Ó'Dúnlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6, 1982.

[59] C. Olson and L. Matthies. Maximum likelihood rover localisation by matching range maps. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'98)*, pages 272–277, Leuven, Belgium, May 1998.

[60] R. Pagnot and P. Grandjean. Fast cross-country navigation on fair terrains. In *Proceedings of the International Conference on Robotics andAutomation*, pages 2593–2598, 1995.

[61] C. Piller. Telecom satellites vulerable to nuclear blast. *Los Angeles Times*, January 17, 1999. Section C (Business).

[62] P. Raeburn and M. Golombek. *Mars: Uncovering the Secrets of the Red Planet.* National Geographical Society, Washington, D. C., 1998.

[63] A. L. Rankin. Path planning and path execution software for an autonomous nonholonomic robot vehicle. Master's thesis, University of Florida, 1993.

[64] N. Rao, S. Kareti, W. Shi, and S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Oak Ridge National Laboratory Document ORNL/TM-12410, July 1993.

[65] E. Rimon and J. Canny. Construction of c-space roadmaps from local sensory data: What should the sensors look for? In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'94)*, 1994.

[66] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, 1992.

[67] H. Rohnert. Shortest paths in the plane with convex polygonal obstacles. *Information Processing Letters*, 23:71–76, 1986.

[68] H. L. Royden. *Real Analysis (Third Edition).* Macmillan Publishing Company, New York, N.Y., 1988.

[69] W. Rudin. *Principles of Mathematical Analysis (Second Edition).* McGraw-Hill Book Company, New York, N.Y., 1964.

[70] H. Saito, T. Adachi, I. Nakatani, and T. Kubota. Autonomous navigation system for planetary rover. In *20th International Symposium on Space Technology and Science*, Gifui, Japan, May 1996.

[71] *Science*, December 5, 1997. Special issue on the Mars Pathfinder Mission.

[72] Z. Shiller, Y.-R. Gwo, and J.-C. Chen. Dynamic motion planning of autonomous vehicles, September 1990. Submitted to *IEEE Transactions on Robotics and Automation*.

[73] T. Simeon and B. Dacre Wright. A practical motion planner for all-terrain mobile robots. Laboratoire d'Automatique et d'Analyse des Systemes, LAAS Report no: 92475, December 1992.

[74] S. Singh and A. J. Kelly. Robot planning in the space of feasible actions: Two examples. Field Robotics Center, Carnegie Mellon University.

[75] M. G. Slack. Situationally driven local navigation for mobile robots. JPL Publication 90-17, Jet Propulsion Laboratory, California Institute of Technology, April 1990.

[76] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. The Robotics Institute, CMU-RI-TR-93-20, August 1993.

[77] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'94)*, May 1994.

[78] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, August 1995.

[79] A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, 2(2), August 1995.

[80] Mars Pathfinder Microrover Flight Experiment Team, 2003/05 Rover Team 2001 Mars Rover Team, and Long Range Science Rover Team. Personal communication, Summer 1996 - Spring 1999.

[81] TRW Space Log 1975. TRW Systems Group, TRW, Inc., Redondo Beach, CA, 1976.

[82] R. Volpe, J. Balaram, T. Ohm, and R. Ivlev. The Rocky7 Mars rover prototype. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems*, 1996.

[83] Voyager mission status, November 17, 1998. JPL Media Relations press release.

[84] R. Welch, January 1999. 2003/2005 Rover Control and Navigation Subsystem Engineer, personal communication.

[85] B. Wilcox, A. Nasif, and R. Welch. Implications of martian rock distributions on rover scaling. In *International Conference on Mobile Planetary Robotics*, Santa Monica, CA, January 29–February 1, 1997.

[86] B. Wilcox, A. Nasif, and R. Welch. A nanorover for Mars. *Space Technology*, 17(3–4):163–172, 1997.

[87] B. Wilcox and T. Nguyen. Sojourner on Mars and lessons learned for future planetary missions. In *Proc. of the 28th International Conference on Environmental Systems (ICES'98)*, Danvers, Mass., July 13–16, 1998. Society of Automotive Engineers (SAE).

[88] C. M. Witkowski. A parallel processor algorithm for robot route planning. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 827–829, Karlsruhe, West Germany, 1983.

[89] A. Yahja, A. Stentz, S. Singh, and B. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'98)*, Leuven, Belgium, May 1998.